PART II
RESEARCH PAPERS

3. INTELLIGENCE OF DISTRIBUTED AGENTS

DISTRIBUTED INTERNET-BASED MULTI-AGENT INTELLIGENT INFRASTRUCTURE SYSTEM

Xiaoli Qin[1] **A.E.Aktan**[2] **Kirk Grimmelsman**[3] **F.N. Catbas**[4] **Raymond Barrish**[5] **M. Pervizpour**[6] **E. Kulcu**[7]
Drexel Intelligent Infrastructure And Transportation Safety Institute
Drexel University
3201 Arch Street, Suite 240
Philadelphia, PA 19104

ABSTRACT

The Commodore Barry Bridge (CBB) is a major long-span, cantilever through truss bridge owned by the Delaware River Port Authority (DRPA). To evaluate the performance of this bridge, it is necessary to implement an appropriate health monitoring system, conduct structural analysis, measure the operating and loading environment as well as the critical responses of the structure. The health monitoring system may be used in order to track operational anomalies, deterioration or damage indicators that may impact service or safety reliability. The knowledge space required to accomplish such complex engineering tasks is innite and uncertain. This engineering domain itself is not well understood. To solve such engineering problems, not only is theoretic knowledge required but also extensive heuristic experience. Organizing and formalizing the theoretical knowledge and heuristic experiences of multidisciplinary human experts is the rst major challenge. The building of an intelligent system that can reason and make rational decisions based on induction/deduction of theoretic knowledge and analysis of heuristic experiences is the second major challenge.

[1]Web: http://www.mcs.drexel.edu/ gxqin; Email: xq22@drexel.edu.
[2]Web: http://www.di3.drexel.edu; Email: Aktan@drexel.edu.
[3]Web: http://www.di3.drexel.edu; Email: grimmeka@drexel.edu.
[4]Web: http://www.di3.drexel.edu; Email: fncatbas@drexel.edu.
[5]Web: http://www.di3.drexel.edu; Email: barrisra@drexel.edu.
[6]Web: http://www.di3.drexel.edu; Email: Mesut.Pervizpour@drexel.edu.
[7]Web: http://www.di3.drexel.edu; Email: Eray.Kulcu@drexel.edu.

This paper presents the writers' progress towards the development of an intelligent infrastructure system that uses integrated technologies of Case-Based Reasoning (CBR) and Rule-Based Reasoning (RBR) to evaluate the performance of the CBB. The system includes a case-base, a rule-base, a CBR agent, a RBR agent and an inter-operational agent. The CBR agent and RBR agent work with both case-base and rule-base. The case-base and rule-base are inter-related through the index schemes. The inter-operational agent evaluates the outputs of the CBR agent and RBR agent to make decisions. This agent can be an alternative human engineer. The CBR methodology is well suited to formulate human experiences and phenomena that would not lend themselves to organization and extraction in terms of rules. In contrast, the theoretical knowledge can be organized using RBR technique. The combination of CBR and RBR technologies offers promise for developing a methodology for solving complex real-life engineering operation problems. The CBR and RBR agents are implemented and wrapped according to CORBA (Common Object Request Broker Architecture) /DCOM (Distributed Component Object Model) standards in order to communicate with each other and an external CORBA server or DCOM server to acquire necessary knowledge.

INTRODUCTION

Case-Based Reasoning (CBR) techniques are a promising for solving many engineering problems. CBR is a subeld of Articial Intelligence (AI) that is premised on the idea that past problem-solving experiences can be reused and learned from in solving new problems. Rule-Based Reasoning (RBR) techniques are commonly used for developing expert systems in terms of building rules for solving generic or specic problems.

This paper discusses the use of combining case-based reasoning and rule-based reasoning techniques to build a multi-reasoning (multi-agent) system to solve a complex domain-specific problem — namely, evaluating bridge performance in civil engineering applications. This paper presents a three-phase approach to building such a system for this domain:

1. *Knowledge Representation for Evaluating Bridge Performance*: building a knowledge-base;
2. *Case-Based Reasoning Engine and Rule-Based Reasoning Engine*: design of the CBR reasoner and intergration of the existing RBR reasoner;
3. *Implementation Issues*: illustrations of how a multi-agent system can be used during the phase of evaluating bridge performance.

Foundation of Case-Based Reasoning and Rule-Based Reasoning Techniques

The *Case-Based Reasoning Cycle* (1) precisely defines a methodology to build a CBR system for a given domain. A case-based reasoning system can be viewed as a model which is a combination of a *case-base* and *knowledge reasoning* process modules. These modules form a *case-based reasoning shell*, also called a *reasoner*. They are the functions used to manipulate the knowledge in the case-base and they act to *process* user inputs, *recall* similar cases, *retrieve* the most similar case, *evaluate and adapt* the retrieved case and update the case memory. The modules interact with the case-base during processing.

Normally, following problems are involved in a CBR system: **knowledge acquisition, knowledge representation, case retrieval, case adaptation** and **the learning mechanism**.

1. **Knowledge acquisition:** How to acquire useful knowledge from application problem domain.
2. **Knowledge representation:** How to use a formal language to represent certain domain knowledge. The knowledge representation theory of case-based reasoning systems primarily concerns how to structure knowledge stored in the case-base to facilitate effective searching, matching, retrieving, adapting and learning. One influential knowledge representation model is the *dynamic memory model* (11). It was developed by Schankand based on his theory, Memory Organization packet (MOP) theory.
3. **Case retrieval:** How to efficiently retrieve from the case-base the case most similar to the current problem. There are two sub-processes involved in case retrieval: one is to retrieve a set of similar cases from case-base, another is to find the most similar case in this set. The first sub-process is accomplished by designing appropriate index scheme for the domain problem. The second task is done using the *Nearest Neighbor Matching Algorithm (NNM)* (7).

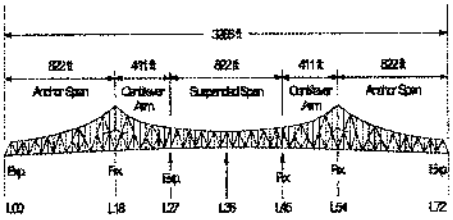


Figure 1. Overview of the Commodore Barry Bridge

4. **Case Adaptation Strategies:** After a CBR system retrieves the most similar case from the case-base, it normally needs to perform adaptation on this retrieved case. There are several adaptation strategies which can be used in a CBR system. They are Simple Substitution, Parameter Adjustment and Constraints Satisfaction (7).
5. **Learning Mechanism:** Learning is the last step in the Case-Based Reasoning system. In a CBR system, after a new problem is solved, the case-base is changed by adding the new case into it. By doing that, the system can retain more and more knowledge along with problem-solving augmentation and achieve learning.

For a RBR system, following problems are involved: **knowledge acquisition, knowledge representation, pattern matching definition** and **execution when pattern matching**. The first two problems have the same characteristics as CBR system. For the next two problems, brief explainations are given below:

1. **Pattern matching definition:** How to find patterns which are stored in rulebase. This is accomplished by implementing Rete matching algorithm which is introduced extensively by Charles Forgy's PhD dissertation
2. **Pattern matching definition:** How to excute actions when RBR inference finds applicable patterns. The inference engine loops through all matched rules and fires exhaustively until no more applicable rules in the rulebase.

Domain Problems and Knowledge Representation

Commodore Barry Bridge. The Commodore Barry Bridge (CBB) is owned by the Delaware River Port Authority (DRPA). It links Chester, Pennsylvania with Bridgeport, New Jersey and was opened to traffic in 1974. The bridge is the 3rd longest cantilever truss bridge in the world with a main span of 1,644 feet and a total bridge length of 13,912 feet. Figure 1 shows the principal structural system of the CBB.

Presently, the Commodore Barry Bridge carries more than 6 million vehicles annually, much of it heavy truck traffic

seeking to avoid the traffic congestion of the busy Philadelphia metropolitan area (3). The bridge owner wished to objectively evaluate this aging and heavily loaded structure.

The Instrumented Monitoring of the Commodore Barry Bridge. The Drexel Intelligent Infrastructure and Transportation Safety Institute (DIII), working in partnership with the DRPA, has been investigating the application of various health monitoring techniques to the CBB. Health monitoring, in the case of civil infrastructure systems, may be considered as measuring and tracking the operating and loading environment of a structure and corresponding structural responses in order to detect and evaluate operational anomalies and deterioration or damage that may impact service or safety reliability. Designing a monitoring system for a long-span bridge was a major challenge for the DIII researchers. In the past two years, DIII researchers have developed and implemented a health monitoring system for the CBB. This system was designed as a first-cut health monitoring system that would measure global and local responses of the structure in critical members and regions of the bridge. The system takes advantage of in excess of 100 data channels to continuously track the loading environment and numerous structural responses of the bridge (3).

The Structural Identification of the Commodore Barry Bridge. A primary step in implementing a successful global health-monitoring system for a bridge is to accurately conceptualize the structural systems. Long-span bridges typically have numerous complex structural details, boundary, movement and continuity systems that require, at the very minimum, identification and understanding from a conceptual perspective in order to design an appropriate health-monitoring system. These systems, when coupled with transient, non-stationary, nonlinear or unknown load effects and responses, create a monitoring situation that is sufficiently complex to justify a conceptualization effort (3).
Conceptualization of the structural systems is most efficiently accomplished through 3D CAD and solid modeling of the structure, site visits, photographs, and heuristics. The Commodore Barry Bridge consists of sixty-three multi-girder approach spans, eleven deck truss approach spans, and a three span cantilevered through truss. The total length of the bridge from abutment to abutment is 13,912 ft (3). Selecting the most appropriate bridge members to monitor was another major challenge for the DIII researchers.
The DIII researchers have conducted extensive studies to identify critical bridge members. Correspondingly, sensors are installed at those critical locations to monitor important parameters. Figure 2 shows locations of critical members and their instrumentation.

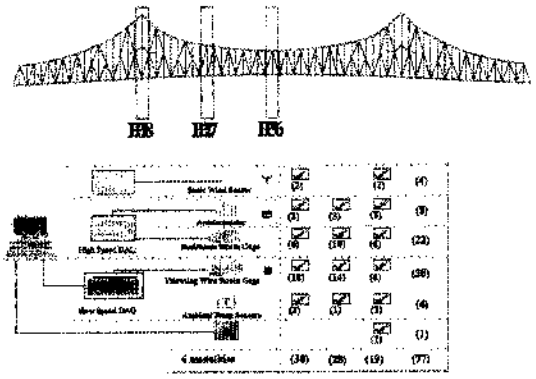


Figure 2. Structural Identification And Instrumentation

Why Is An Intelligent Reasoning System Necessary for Health Monitoring? In health monitoring of large structural or infrastructure systems which have the probability of brittle failure modes, engineers are very interested in "intelligent sentries". In the case of the CBB, the researchers have developed sensor systems to monitor the local conditions at the critical regions that are susceptible to fatigue cracking. If these systems sense an incipient cracking, they should inform a human. In this type of effort, a false positive event is very dangerous while a false negative event is totally unacceptable. Therefore, the intelligent agent should be able to follow redundant reasoning and fusion of data from various sensors to rule against false positive while being cognizant of false negative.
The second reason an intelligent agent is needed is for detecting and interpreting the initiation of conditions favorable to deterioration. For example, analysis of internal humidity and electro-chemical characteristics for concrete elements could establish the onset of reinforcing steel corrosion.
Since a health monitor or supervisory control and data acquisition (SCADA) system for a major bridge or a major infrastructure system must utilize many sensors distributed over a large geometric domain, it is impossible for humans to continuously watch for continuously watch for incidents, events and complex phenomena pointing to out-of-ordinary incidents, events and complex phenomena pointing to out-of-ordinary conditions with the structure. The only way a SCADA can become completely effective is if it has self-intelligence to alert human managers when needed.

Multi-Disciplinary Research. The monitor system for CBB has been functioning since 1998, and additional data has been obtained by many controlled tests. Data has been interpreted by the researchers for characterizing the mechanical characteristics and the loading and response environment of the bridge structure in terms of a 3D finite-element

model. Researchers continue recording and viewing data from continuous measurements in real-time, and from controlled load tests and ambient vibration tests that are conducted intermittently. The data is used for calibrating the analytical model and validating its reliability for simulating phenomena at the regional and element levels (2), (4), (3), (8).

Research at DIII is conducted in three distinct areas. The first research direction involves investigating, designing and implementing health monitoring systems for civil infrastructure systems. This research is primarily conducted by civil and electrical engineers. The second research area involves structural identification and analysis of instrumented civil infrastructure systems. This requires a team of civil, mechanical, and electrical engineers. The third research area focuses on intergration. This research takes advantage of computer science techniques to fuse distrubuted applications. In addition, knowledge engineering methodology is used to compile, structure and model human knowledge to solve complicated civil infrastructure problems. This research requires the efforts of a computer software engineer. This paper discusses some of DIII's efforts in the third research area.

Knowledge Representation of the CBB . Because of the inherent complexity of the CBB bridge project, the knowledge space in this domain is incomplete and dynamic. It emcompasses civil engineering, electrical engineering and computer science. It is not practical to fully compile and model the knowledge in this project domain. However, acquiring and modeling the primary knowledge for major compents of the project from human engineer is approachable. The major components of CBB project include health monitoring instrumentation and structural analysis. In this paper, a fragment of the knowledge for structural analysis of the CBB and knowledge representation of it is presented. Knowledge acquisition is achieved by specifying only the important features of the problem. Features are only collected if they help solve the specific problem. Other knowledge that is not directly related to solving the problem is discarded. In this approach, a set of important features is predefined for the problem, and knowledge acquisition is done manually by a knowledge engineer. Because of restrictions mentioned above, the system will have some limitations. These limitations will be briefly discussed in the last section

The researchers have conducted extensive research on Case-Based Reasoning (CBR) and Rule-Based Reasoning (RBR) methodologies. Both of these methods provide a very promising way to organize, construct and program human knowledge into a system. This system can contain human experience, theroretical knowledge and respond to the real world based on build-in reasoning mechanisms.

A language called CASL (5) is used to represent knowledge pertaining to CBB project in this case-base. The structure of the case-base is based on Memory Organization Packet (MOP) theory (11).

A language called CLIPS (9) is used to build the rule-base. This language provides three paradigms to organize knowledge which are rule-based, object-oriented and procedure-based.

Case-Based Reasoning and Rule-Based Reasoning Engines

A reasoning engine is software agent which perceives knowledge from the knowledge base, conducts logic inference and reasoning and concludes results. In general, a reasoning engine is used to reason on a specific kind of knowledge base. For example, CLIPS (9) is a tool which provides language used to build a rule-base and a rule-based inference engine is used to reason on the rule-base built by this language.

The case-based reasoning engine is the reasoning system which allows a researcher to use archived cases to solve domain problems. Once domain knowledge has been used to build the case-base, organize memory, build indices, etc., the reasoning engine can execute searches based on the index scheme. The engine can also perform other reasoning processes, including case retrieval, adaptation and system learning.

The rule-based reasoning engine is the reasoning system which reasons on a rule-base. The domain knowledge is compiled, modeled and structured in terms of a series of rules. The rule-based reasoning engine automatically matches facts against patterns and determines which rules are applicable. If they are, the engine performs certain actions specified by knowledge base.

KNOWLEDGE REPRESENTATION FOR CBB BRIDGE PERFORMANCE EVALUATION

Problem Formulation

The CBB project contains two major components. One of them is structural identification and analysis of the bridge. The other is health monitoring of the bridge. In this paper, only a small segment of the domain problem related to the former will be presented as an example. The objective of the structural identification approach is to characterize the as-is structural condition and the loading environment of a bridge through experimental information and analytical modeling. Experimental data acquired from instrumentation is com-

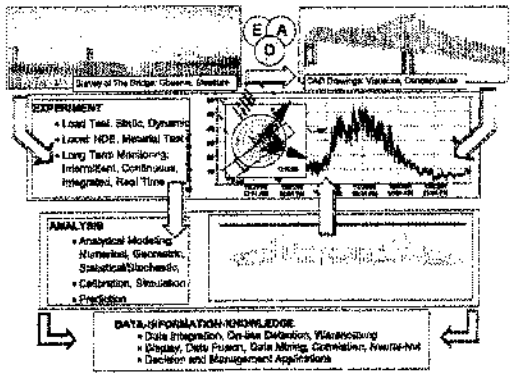


Figure 3. Structural And Loading System Identification

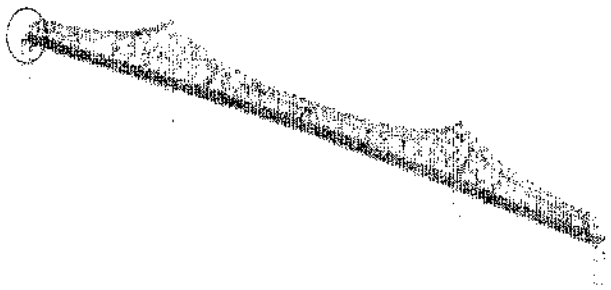


Figure 4. Finite Element Model of the Commodore Barry Bridge

pared to results obtained from an analytical model of the bridge. The results can be used to evaluate the correctness and reliability of the analytical model, to evaluate the performance critical bridge elements, and to issue notifications regarding the safety of structures.

Several 3D Finite Element (FE) models of the CBB were developed to serve as tools for engineering decisions. The analytical models incorporate the contribution of all force resisting elements and mechanisms, particularly those associated with out-of-plane elements, into the analytical model. In this manner, these elements and mechanisms can contribute to the behavior of the model as they do in the actual structure, enabling more realistic and accurate simulations of retrofits, modifications, and loading scenarios (4). The FE models were developed in several stages. First, the structural systems of the bridge were conceptualized by review the design calculations and drawings, shop drawings and site visit. Second, the structure was re-constructed using a 3D CAD model. Finally, the CAD model was transformed into a FE model using a commercial software program. Figure 3 summarizes the development stages and Figure 4 shows the complete 3D FE model of the through truss structure.

DIII researchers conducted a controlled load on the bridge to measure the critical responses. The measured responses provide information necessary to verify analytical models of the structure. The controlled load test on Commodore Barry Bridge consisted of a static load test at pre-identified locations and a crawl speed test. The Commodore Barry Bridge was loaded statically by positioning two large cranes in various configurations. The measured responses included strain measurements for vertical truss members, lower chord truss members, upper chord truss members, floor beams, and for deck stringers near the hangers and midspan regions of the through truss.

The bridge member responses at several locations were measured under a 108 kip crane loading for several loading configurations. The loading configurations were also simulated in the finite element model to obtain analytical responses, which were then compared with the experimental results. The finite element model was found to represent the measured response of the bridge quite well. Examples of model validation and calibration are given in this section along with examples of measurements that illustrate the complexities associated with the response of the bridge. The maximum incremental strain in one hanger due to the prescribed loading condition was measured to be 43.23 microstrain (1.25 ksi). After simulating the same loading condition in the model, the strain value was obtained to be 45.5 microstain. Some formula related above result are given as follows:

$$\sigma = \frac{N}{A}$$

and

$$\varepsilon = \frac{\sigma}{E}$$

where

- σ: Stress (ksi)
- ε: Strain ($\frac{in}{in}$)
- A: Cross sectional area of hanger (in^2)
- N: Axial load (kips)
- E: Young's modulus (ksi)

In the following sections, the hanger analysis of Commodore Bridge will serve as an example to show how to represent the knowledge related to this analysis problem. How to reason knowledge pertaining to it using multi-agent inference engines will also be discussed below.

The CBR Representation Schema

The knowledge pertaining to CBB project can be represented in any kind of representing language. The reason that Case-Based Reasoning Language (5)(CASL) is chosen to represent our domain knowledge is because the knowledge associated problem domain is extremely dynamic and uncertain. Tremendous heuristic experiences are needed to solve practical problems. CASL is a language specially good for model and structure heuristic experiences. The contents of a case-base are described in a file known as a case file, using the language CASL. The reasoner uses this case file to create a case-base in the computer's memory, which can then be accessed and adapted in order to solve problems using Case-Based Reasoning mechanism.
Like any other representing language, CASL has strict syntax, semantics, keywords and operators. The syntax of CASL specifies the grammar rules of organizing knowledge, and the semantics of CASL give the concise interpretation of a sentence written in CASL with correct grammar. CASL defines some basic types in the language: identifiers, strings, numbers and operators, etc..

CASL normally divides a case-base into several modules, each of which has its own syntax features and semantic explanations.
CASL semantics define the meaning of a sentence by specifying the interpretation of the keywords and basic types, and specifying the meanings of operators. In the syntax blocks of CASL, all keywords and literals are given in bold type.
A small example about hanger analysis is provided to show how to use CASL to represent domain knowledge. When a problem is presented, certain conditions are specified. These specifications are the input to the problem solver, or *CBR reasoner*. The CASL structures the knowledge about input problems by defining the primary features of a problem. Every primary feature has a weight associating to it. This weight vaule indicates the importance of this feature.
The brief explanations of primary modules and examples are given below:

1. Introduction. This module defines introductory text which is displayed when the reasoning process (reasoner) is run. The purpose of the text is to help the user understand the contents of the case-base or anything else of note.

2. The Case Definition. The purpose of this block is to define the problem features contained in a case.
In the hanger analysis problem, the most important features are axial forces and bridge type. These features' weight values are set to be 5 (reference weight). The cross sectional area of hanger and Young's modulus etc., are not that important, comparatively speaking. Therefore, their weight values are set to be 0 (reference weight). A sample case definition using CASL is given below:

```
case definition is
field axial-force type is (number) weight is 5;

field bridge-type type is (Long-Span (Suspension, Cable-
Stayed, Truss, Arch), Short-Span, Culvert) weight is 5;

field axial-force type is (number) weight is 5;
field cross-section type is (number) weight is 0;
field Youngs-modulus type is number weight is 0;
field Experimental Data type is number weight is 0;
end;
```

3. Index Definition. The purpose of this module is to define which fields are to be used as indices.
This part defines the fields which are used as indices when searching for a matching case. The index scheme defines the methods by which the reasoner should access the case memory. Indices are intended to streamline the matching process. The index features are parts of the new problem specification. For example, we use the features *axial-force* and *bridge-type* as main indices to search the knowledge-base. The sample representation is given below:

```
index definition is
index on axial-force;
index on bridge-type;
```

4. The Adaptation Rule Definition. The purpose of this block is to define rules used to modify a retrieved case from the case-base to make it fit the current problem specifications. The *global repair rule definition* defined in this module allows adaptation rules to be applied on any modified case. The rules defined here are derived from domain knowledge, formulae and constraints.
When the old hanger analysis whose "description of problem definition" part is the most "similar" to the current problem definition is retrieved from the case-base, its solution part must be modified to fit the current problem definition. The reasoner performs adaptations to an old solution according to certain rules defined by domain experts. The **repair rule definition is** block of CASL can be used to define those rules. In the hanger analysis problem, the following rules (strategies) are defined:

(a) Perform simple parameter substitution: substitute parameters of old problem definition into new user input.

(b) Perform old solution adjustment to make it fit substituted user input (current problem) according to domain formulae.
(c) Check global constraints defined in the case-base to guarantee that no conflicts result.

In the sample given in Algorithm 1, the *change_value_1* is an adaptation rule. It tests a certain condition (represented by a formula) first; when the condition is satisfied, the action is fired.

Algorithm 1: Adaptation knowledge representation:

```
(1)   repair rule definition is
(2)   repair rule change_value_1 is
(3)   when
(4)   axial-force ≥ radial-force
(5)   then
(6)   evaluate Stress Of Hanger σ to N/A
(7)   evaluate Strain Of Hanger ε to σ/A
(8)   repair;
(9)   end;
(10)  end;
```

5. Case Instance Definition. The purpose of this block is to define the structure of a case instance. A case must contain two parts: the problem part and the solution part. The *local repair rule definition* defined in this module allows adaptation rules to be associated with a case. These rules are invoked after the *global* adaptations have run their course. The past experiences of hanger analysis for applications are stored in the case-base. Representation of these experiences requires the design of certain structures which can represent cases properly. Normally, an experience (case) includes a problem statement part and a solution part. The **case instance is** block of CASL provides a kind of structure and function. This block defines the same structure of problem statement as the **case definition is** block defines.
A sample representation of a case is Algorithm 2:

Algorithm 2: Case Instance Representation:

```
(1)   case instance Hanger Analysis is
(2)   bridge-type = Truss;
(3)   axial-load = N;
(4)   cross-section = A;
(5)   Young-Modulus = E;
(6)   Experimenta Data = D;
(7)   solution is
(8)   Stress = S1;
(9)   Strain = S2;
(10)  permissable capacity= C;
(11)  local repair rule definition is
(12)  repair rule rule_1 is
(13)  when
(14)  bridge-type ≠ Truss
(15)  then
(16)  pr 'Abandon your selection ! ';
(17)  pr 'This case can not be repaired to let you use!';
(18)  reselect;
(19)  repair;
(20)  end;
(21)  end;
```

The RBR Representation Schema

CLIPS (C Language Integrated Production System) is an expert system tool developed by the Software Technology Branch (STB), NASA/Johnson Space Center (9) . It is designed to facilitate the development of software to model human knowledge or expertise. There are three ways to represent knowledge using CLIPS in a rulebase:

(a) **Rule-Based Knowledge Representation:** In this paradigm, knowledge is represented as a series rules. Rules are used to represent heuristics which specify a set of actions to be performed for a given situation. A rule is composed of a *if* part and *then* part. The *if* part is a set of patterns which which specify the facts which cause the rule to be applicable. The process of matching facts to patterns is called pattern matching (9). The built-in inference engine matches facts against patterns and determines which rules are applicable.
(b) **Object-Oriented Knowledge Representation:** In this paradigm, knowledge is represented as a series modular components which inherit object-oriented mechanism. For example, this mechanism makes hierarchy knowledge models possible.
(c) **Procedural Knowledge representation:** In this paradigm, knowledge is represented in terms of procedural style like conventional language C, C++ and Pasal etc. This capability is extremely useful when knowledge can not be represented using rules or object-oriented mechanism.

Example of Rule-Based Knowledge Representation. The following example shows how a rule-based representation is used for the CBB hanger analysis. The example shows that when the RBR inference engine finds experimental data from instrumented hanger that is much larger than

analytical data from FE model, it fires and generates warning signal.

Algorithm 3: Rule-based representation by CLIPS:
(1) **(defrule** *Warning-Signal*)
(2) *(experimental-data-isData)*
(3) *(analysis-data-isA-Data)*
(4) **assert**(*Something wrong on the CBB bridge!*)

MULTI-AGENT REASONING SYSTEM

System Overview

The *case-based reasoning engine*, also called *reasoner*, takes problem specifications and a case-base file as its inputs, performs reasoning about the problem, and returns an answer to the user automatically. The reasoning engine of a case-based system consists of four process modules; each of those modules performs certain functions. The modules interact with the case-base and form a reasoning cycle. The first module, *Retrieved case*, takes the current problem specifications as input and outputs a retrieved case. The second module, *Solved case*, decides whether a retrieved case needs to be adapted. This module either returns to the user a solution without further modification or passes a solution to the next module which will perform adaptation on the case. The third module, *Repaired case*, performs this adaptation and returns an adapted case to the next module. The fourth module, *Learned case*, decides whether this new resolved case needs to be stored in the case-base. The kernel of CBR engine used in the problem domain was developed by Center for Intelligent System. University of Wales (5). The primary author has developed a wrapper for this engine, added extra features for this engine and has added extra features including a Graphical User Interface (10).
The *rule-based reasoning engine* which is part of the CLIPS system, also called inference engine, was developed by NASA's Johnson Space Center (9). The CLIPS was a *forward chaining rule* language based on the *Rete pattern matching* algorithm. The inference engine was implemented as different modules. Every module has different functions and purposes. Detailed information about these modules is provided at the CLIPS website (9). Only basic ideas of this inference engine are introduced here. When the inference engine is invoked (perceives input from outside world), it automatically looks at the rulebase and matches facts against patterns which are defined by the knowledge engineer. It then determines which rules are applicable. It selects a rule

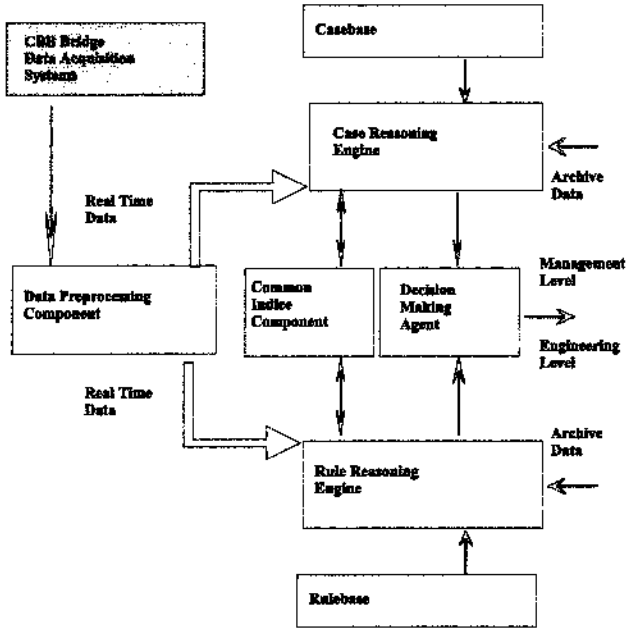


Figure 5. Multi-Agent Reasoning System Architecture

and then the actions of the selected rule are executed. The inference engine then selects another rule and executes its actions. This process continues until no applicable rules remain (9).
The following sections present how the writers will integrate and implement these modules and the RBR inference engine. Only a detailed introduction of CBR reasoning engine is presented here.

System Architechures

Main System Architecture. Figure 5 shows the architechture of multi-agent reasoning system.
There are seven modules in the system. Each module performs specific tasks and functions.

(a) **CBB Bridge Data Acquisition Module:** This module collects data from instrumented sensors on the CBB bridge and performs buffering and raw data storage functions. The module consists of several different data acquisition hardware and software systems acquired from a variety of vendors. Some of the vendors provide built-in functions which can be integrated with commercial data processing software. This makes communication between the *CBB Bridge Data Acquisition Module* and the *Data Preprocessing Component* discussed below possible. For example, the OPTIM Electronics data acquisition system (6) provides

OLE interface that enables the system to communicate with the commercial data processing software LabView which also supports the OLE standard.

(b) **Data Preprocessing Component:** This component performs data preprocessing. It takes the *CBB Bridge Data Acquisition Module* as input, checks data quality, eliminates electrical signal errors and conducts preliminary data analysis and other related tasks. This module's kernel could be some data processing software like LabView. The module outputs data either immediately to a display through some interface like web browser or automatically to a database. For example, the former can be implemented through LabView that can read data from OPTIM and send it to a web browser for direct and immediate display to the user. The latter can be implemented by writing a customized program which reads data from LabView's buffer or from its data storage disk. The program then routes the data to an archived database.

(c) **Case-Based Reasoning Engine:** This module performs CBR reasoning. It perceives knowledge from casebase, extracts data from both the *Data Preprocessing Component* , an archived database and the *Common Indice Component*. It performs reasoning based on all the information mentioned above and returns reasoning results to the *Decision Making Agent*.

(d) **Rule-Based Inference Engine:** This module performs RBR reasoning. It perceives knowledge from the rulebase, extracts data from both the *Data Preprocessing Component* , an archived database and *Common Indice Component*. It performs reasoning based on all the information mentioned above and returns reasoning results to *Decision Making Agent* which will be introduced below. This module will be presented in more detail later. .

(e) **Common Indice Component:** This component serves as hub to connect CBR system and RBR system together. It takes the user's problem as input, checks important features of domain problem and outputs these features to the CBR engine and the RBR engine. It triggers both engines through industry standard protocols such as COM(DCOM) which allows distributed applications to communicate with each other. Since the source code for the CBR and RBR engines are publicly available and they were created using C language, it is not difficult to program a wrapper with a standard interface for both engines.

(f) **Decision Making Component:** This component collects the output from the CBR reasoning engine and the RBR inference engine. The component can be a software entity or a human entity. The entity judges the output from both engines and conducts reasonable

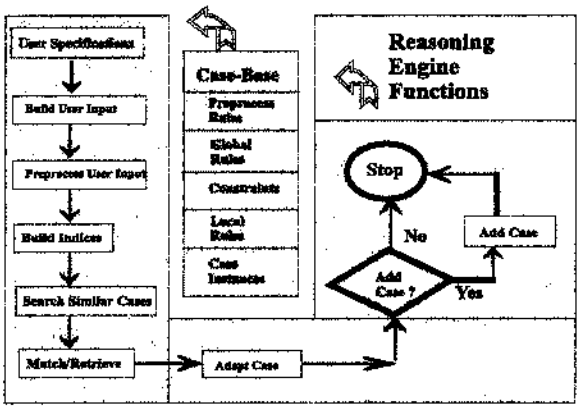


Figure 6. The primary functions of a CBR Reasoning engine

actions. For example, a software entity can issue warnings based on its judge from outputs of inference engines to warn the human manager that some members of the bridge need to be reinforced or that the bridge should be closed due to some catastrophic event.

Reasong Process of CBR Engine

The flow-chart in Figure 6 shows the main algorithm behind the implementation of CBR reasoning engine. The two hollow arrows in the figure illustrate that the reasoning engine must interact with the case-base.

The flow-chart shows that the requirements of a module can be broken into pieces or procedures called by the main function. It also shows that a CBR engine forms a reasoning loop. This reasoning loop begins with the procedure *User Specification* and ends with the procedure *Add Case*. Primary procedures used in the main algorithm are discussed below separately.

Building the Index

The performance of a CBR system is determined by the CBR reasoning engine whose efficiency is in turn determined by the design of the *index scheme* and the *case-base memory organization*. The index scheme design includes how to specify index features and how to build them in computer memory. The index features are set by domain experts and are represented by the block **index definition is** of CASL. The procedure *Build Indices* takes the representations of index features as input and uses these to build the index scheme. A *linked-list* data structure was chosen to hold the index feature input. The procedure *Build Indices* places all the index features into the linked-list, and at the same time, builds the case-base memory organization. Figure 7 illustrates these ideas.

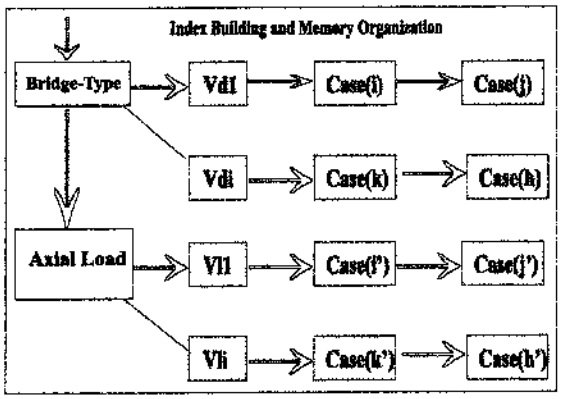


Figure 7. The index building and case-base memory organization

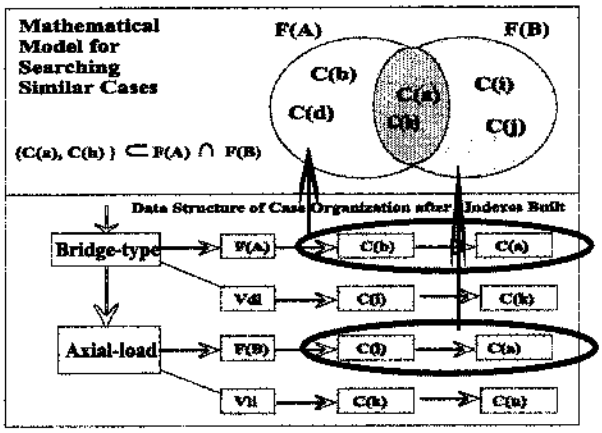


Figure 8. The mathematical model and an example for searching similar cases

In this CBR system for the hanger analysis problem, two features have been specified as index features: *bridge-type* and *axial-load*. Each index feature is a node of the linked-list and the data type for the nodes is the **struct** type in C. The fields of the **struct** are used to hold attributes of the index features. Figure 7 shows this data structure for the index features and case-base memory. The procedure *Build Index* first links the index features *shaft diameter* and *load direction*. It then checks every attribute of the index features. For each attribute, *Build Index* searches for all the cases with the same attribute value in the case-base file and links all of these together.

Case Matching, Ranking and Retrieving

The purpose of building an index scheme is to speed up *searching*. Here, *searching* means to find a set of cases from the case-base which are similar to the current input case. However, the goal is to find the case that has the maximum similarity to the input case. Thus, a mechanism to rank the similarity of cases is needed. In this section, the procedure necessary to accomplish two goals (finding a similar case set and finding the most similar case in this set) is discussed.

First, a mathematical model is presented to demonstrate how to find a set of similar cases in the case-base. What are *similar cases*? **Given an input case with certain index features and their attributes, similar cases are those cases whose index features and attributes are exactly the same as the corresponding input case's.** Figure 8 shows these ideas.

The top portion of the Figure 8 illustrates the mathematical model for finding similar cases. The left and right circles represent attributes $F(A)$ and $F(B)$ of index features A and B of an input case respectively. The $C(n)$ represents a case n. If the left circle includes $C(b), C(d), C(h)$ and $C(a)$, which are the cases with attribute $F(A)$ of feature A, and the right circle includes $C(i), C(j), C(a)$ and $C(h)$, which are the cases with attribute $F(B)$ of feature B, then their intersection contains cases $C(a)$ and $C(h)$, which have both attribute $F(A)$ and $F(B)$. This can be represented in set theory as:

$$\{C(a), C(h)\} \subset F(A) \cap F(B)$$

The bottom portion of Figure 8 provides a corresponding example to illustrates how this process occurs in the case-base.

After all similar cases are found, a mechanism to find the most similar case in this set is needed. In the system, the **Nearest Neighbor Matching algorithm (NNM)** (7). Figure 9 illustrates how this algorithm works in the CBR system for the hanger analysis. To simplify the discussion, it is assumed that all of the component loads applied to the hangers are at in the same direction.

The basic idea of the NNM algorithm is to compare the attribute value of each feature for each case in the set of similar cases to every corresponding feature's attribute of the input case, calculate the comparison values and then sum them for each case to get a total comparison value.

In the upper portion of Figure 9, the circles represent cases, the dots represent attribute values of features, index i represents the input case, and index j represents cases in the set of similar cases. The index k represents the features in a case. The case A and case B in the figure are the cases from the similar cases' set. The function $d(k)(ij)$ represents the attribute's comparison value of one of the features (feature k) between the input case and case A, which is equivalent to

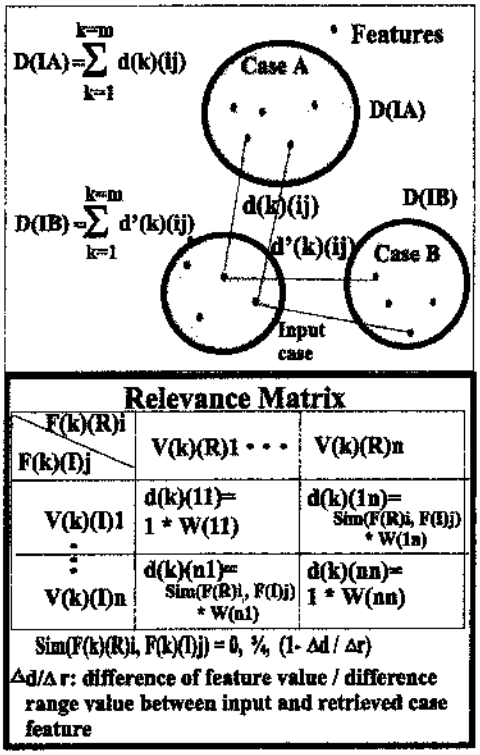


Figure 9. The Nearest Neighbor Matching algorithm

the following formula (7):

$$W(ij) * Sim(F(k)(R)i, F(k)(I)j)$$

where:
k: a feature of a case.
$W(ij)$: the weight of a feature, defined in the case-base file.
$Sim(F(k)(R)i, F(k)(I)j)$: the degree of similarity between one of the features in the input case and the corresponding feature in a case from the similar case set.
The total attributes' comparison value for a case is D(k)(IA), which is equal to the numeric function

$$\sum_{k=1}^{n} W(ij) * Sim(F(k)(R)i, F(k)(I)j)$$

After finishing all calculations, the NNM algorithm selects the case which has the highest value of $D(k)(ij)$ to be the most similar case.
The key component of the NNM algorithm is the calculation of an attribute's comparison value for a feature between a similar case and the input case. A matrix called the *relevance matrix*, shown in the lower part of Figure 9, is used to explain how to calculate every feature's attribute comparison value. In the matrix, $F(k)(R)i$ means "the feature k of a case from the similar case set which has possible attribute i, where the range of i can be from 1 to some finite number". $F(k)(I)j$ has a similar meaning except in reference to the input case. So, the first row of the matrix represents all the possible attributes of feature k of a similar case, and the first column represents all the possible attributes of feature k of the input case. The intersection of row and column is the comparison value of the feature k. The $W(ij)$ is the weight of a feature in a similar case. The degree of similarity $Sim(F(k)(R)i, F(k)(I)j)$ has three possible values. First, if two features match exactly, the degree of similarity equals 1. Second, if two *abstract symbols* are similar, its value is $\frac{3}{4}$. Third, if two *numbers* are similar (i.e., both fall within the range defined in the *modification* block), then a value is calculated which reflects how close they are in proportion to the range. Then, the $Sim(F(k)(R)i, F(k)(I)j)$ can be calculated by:

$$1 - \frac{\Delta d}{\Delta r}$$

where: Δd is the difference of the feature values between the input case and the retrieved case and Δr is the difference range value. For example, if the attribute value of feature *axial load* for the input case is 64 kips, and the corresponding value for a similar case is 84 kips Newtons, then Δd = 84-64=20. If the definition for the range of similarity is from 44 to 94, then Δr = 94-44=50. *Similarity between 64(input) and 84(a similar case)* $= 1 - \frac{20}{50} = 0.6$
Algorithm 4 defines the functions needed to find similar cases and the most similar case as mentioned above. The procedure *Index_List_Searching( )* performs searching on the linked-list of index features. The procedure *Case_List_Searching( )* searches out cases whose attribute value for certain features is the same as the input case's. The procedure *Computing_Weight_Cases( )* performs calculates the weight of a retrieved case and returns this value. The procedure *Evaluating_Similar_Cases( )* ranks a case with a weight. The procedure *Retrieving_Heaviest_Case( )* retrieves the case with the highest rank and returns this.

Adaptation of Cases

It is rare for a retrieved case to be exactly the same as the newly defined problem. Most of the time the retrieved case is only a similar situation, and so problem definitions and corresponding solutions must be modified so that the modified case fully fits the current situation and its solution fully

Algorithm 4: Case matching, ranking and retrieving:
Input: User's input problem specification.
Output: The retrieved case with highest weight.

```
MATCHING_RANKING_RETRIEVING(UserInput)
(1)   begin
(2)   while true
(3)      do
(4)      Index_List_Searching( );
(5)      Case_List_Searching( );
(6)      Computing_Weight_Cases( );
(7)      if Case_Matching_Exact = True;
(8)         return Retrieving_Case();
(9)      else
(10)        Evaluating_Similar_Cases( );
(11)        Retrieving_Heaviest_Case( );
(12)        end
```

satisfies the current problem requirements. This procedure as a whole is called the case adaptation (repair) process. A series of rules are defined for adapting cases. These rules are provided by domain experts or domain axioms and are applied to each case whenever it is necessary.

Adaptation rules are divided into *global rules* and *local rules* The reasoner uses *global rules* to examine the problem fields and solution fields of the retrieved case. These rules are also used to adapt the parameters of the retrieved case and check constraint satisfaction conditions which are specified by the knowledge-base. If there are any constraint conflicts, the repair rules provide a new problem-solving proposal. Otherwise, they adapt the solution of the retrieved case to the new problem. Sample adaptation rules for global repair are described in **Algorithm 1**.

Figure 10 shows that a linked-list data structure is used to store these adaptation rules. In the figure, every node has two fields: one stores the condition of a rule, the other stores the action. The procedure given in **Algorithm 5** scans the rule list repeatedly as it performs adaptation on a retrieved case; if the condition part is true, it executes the corresponding actions on the case.

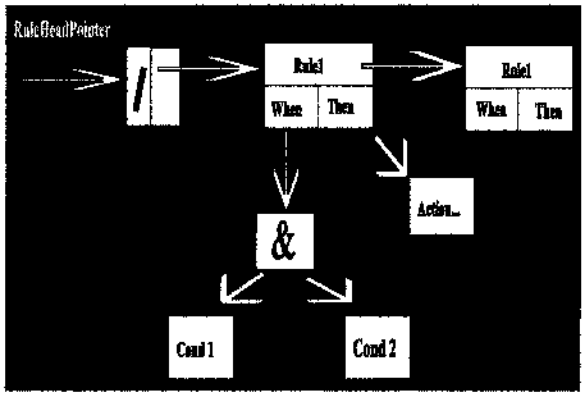


Figure 10. The data structure of global and local rules

Algorithm 5: Algorithm for case adaptation:
Input: Retrieved case.
Output: The modified case.

```
CASE_ADAPTATION(RetrievedCase)
(1)   begin
(2)   while true
(3)      do
(4)      if Global_Rules = True;
(5)         Finding_Global_Rule_Headpointer( );
(6)         Searching_Global_Rules( );
(7)         Apply_Modifying_Retrieved_Case( );
(8)         Parametric_Adaptation( );
(9)         Constraints_Adaptation( );
(10)        Evaluating_Solutions( );
(11)        return Modified_Satisfied_Case;
(12)        end
```

FUTURE WORK

This research touched upon both AI/CBR/RBR and bridge engineering domains. The system discussed can also serve as a template for other engineering domains. The following areas are envisioned for future research.

(a) **Implementation issues:** Several challenges must be overcome in order to implement a multi-agent system. The first challenge relates to knowledge engineering problems which are discussed below. Another challenge arises from the fact that the system is composed of different modules, components and applications which are eventually distributed on different computers and on a network. Developing efficient interfaces and a wrapper to permit them to effectively communicate with each other is a major hurdle.

(b) **Knowledge engineering issues:** Because of the limitations of the CASL and CLIPS used to build our system, there are still many limitations in expressing problem solving intent. The case collection process is quite complicated and inefficient, and case-base maintenance is very unstructured. This makes debugging the case-base very difficult. The rule scope defined for the project domain is extensive because of the complexity of the CBB project. Improved methodologies for case collection, rule collection and better protocols

to maintain the case-base and rule-base are needed.

(c) **Knowledge acquisition issues:** The authors built attribute (features) pairs during the initial design to allow the user to interactively input knowledge. If the system is expanded (especially cross-domain), it would be very difficult to enumerate all the features during design time to cover any and all possible problem specifications. Therefore, the development of an autonomous knowledge acquisition system is a future challenge.

(d) **Indexing issues:** The authors built a fixed feature-based index scheme during the initial design to speed up searching. However, as stated above, if the system is expanded, it would be impossible to optimize this choice of index features. as the system is utilized, many additional features may become important primary design factors. Since these features are not initially coded into the case-base or rule-base, the system will fail to find cases or match rules which have these important features. Developing a dynamic index scheme that will address this situation is an additional research need.

(e) **Graphical reasoning issues:** In the Commodore Barry Bridge project domain, many problems are solved by heuristic experience. Such experience is routinely relied on for interpreting processed images or graphics; therefore, a graphical inference capability becomes necessary. How to combine textual reasoning procedures with graphical reasoning procedures is another very important issue.

CONCLUSIONS

This paper discussed a system that uses Case-Based Reasoning and Rule-Based Reasoning as both a cognitive model and problem solving methodology to deal with a bridge engineering problem for civil engineering applications. The authors believe that this work will produce several insights into how AI, CBR and RBR techniques can be better applied to more realistic engineering problems:

(a) **Knowledge Capture:** Because the knowledge space for the Commodore Barry Bridge domain is extremely incomplete and dynamic, it is difficult to strictly rely on formalizing general, *a priori*, rules to help engineers to solve problems or automate the problem solving process. But using CBR techniques, the extensive experience of many experts can be stored in a case library. In contrast, rule-based techniques can compensate for the shortcomings of case-based techniques. Through exploration of a useful rule-based tool like CLIPS, the knowledge can be modeled using object-oriented mechanism.

(b) **Adaptability:** CBR techniques can integrate knowledge acquisition, reasoning mechanisms, knowledge storage and learning in one platform. Therefore, a system using CBR techniques can possibly grow and expand to encompass a wider variety of assemblies without changing the fundamental system structure.

(c) **Augmenting Intelligence:** The proposed system, rather than being completely autonomous, interacts with the user to obtain knowledge. It provides the flexibility to draw conclusions either from the system itself automatically or by allowing the human engineer to decide which actions he/she should take.

(d) **Human-Guided Search:** The system also provides the flexibility to allow the engineer to loosen index constraints to continue reasoning when an exact search fails. In this manner, the engineer has the most opportunities to obtain a solution that is useful for his/her current problem.

Acknowledgements. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of the supporting government and corporate organizations. The authors gratefully acknowledge research support from the following individuals: Mr. Robert Box (DRPA), Drs. Steven Chase and Ghasemi (FHWA), and Drs. S.C. Liu and Ken Chong (NSF).

REFERENCES

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7:39–59, 1994.

[2] A. Emin Aktan, Kirk A. Grimmelsman, Raymond A. Barrish, and F.N. Catbas. Structural identification of a long span bridge. *Proceedings of the Fifth International Bridge Engineering Conference*, 1, 2000.

[3] Raymond A. Barrish, Kirk A. Grimmelsman, and A. Emin Aktan. Instrumented monitoring of the commodore barry bridge. *Proceedings of SPIE Fifth International Symposium on Nondestructive and Health Monitoring of Aging Infrastructure*, 2000.

[4] F.N. Catbas, Kirk A. Grimmelsman, and A. Emin Aktan. Structural identification of commodore barry bridge. *Proceedings of SPIE Fifth International Symposium on Nondestructive and Health Monitoring of Aging Infrastructure*, 2000.

[5] Center for Intelligent System. University of Wales. http://www.aber.ac.uk/ dcswww/Research/arg/cbrprojects/getting_caspian.html. Caspian.

[6] OPTIM Electronics http://www.optimelectronics.com, 2000.

[7] Janet Kolodner. *Case-based Reasoning*. Morgan Kaufmann Publishers, Inc., San mateo, CA 94403, 1993.

[8] Eray Kulcu, Xiaoli Qin, Raymond A. Barrish, and A. Emin Aktan. Information technology and data management issues for health monitoring of commodore barry bridge. *Proceedings of SPIE Fifth International Symposium on Nondestructive and Health Monitoring of Aging Infrastructure*, 2000.

[9] NASA. Clips. http://www.ghg.net/clips/clips.html, 1999.

[10] Xiaoli Qin and William C. Regli. Applying case-based reasoning to mechanical bearing design. *2000 ASME International Design Engineering Technical Conferences and the Computers and Information in Engineering Conference*.

[11] C.K. Riesbeck and R.S. Schank. *Inside case-based reasoning*. Erlbaum, Northvale, NJ, 1989.

Stigmergy – An Intelligence Metric for Emergent Distributed Behaviors

Richard R. Brooks
Applied Research Laboratory
The Pennsylvania State University
P.O. Box 30
State College, PA 16803-0030
Email: rrb@acm.org

Abstract

Individual autonomous components can be constructed using simple behaviors based entirely on locally available information. Simple components aggregate to form complex systems with complex behaviors. Artificial life research has proposed guidelines for constructing colonies of autonomous systems. Simulations mimicking biological systems show these guidelines adequately explain the behavior of many insect species. The complexity of aggregated behavior often depends on *stigmergy*. Stigmergy occurs when behaviors by individuals modify the environment while being regulated by the environment's state. Stigmergy has generally been studied for the *forward problem*: predicting the consequences of local behaviors. It is also applicable to the *backward problem*: synthesizing local behaviors to fulfill a global need. The concept provides an objective measure of intelligence for natural and synthetic systems. A system's intelligence is measured by its amount of effective stigmergy. It not only adapts to a changing environment, but also modifies the environment to suit the system's needs and goals.

Keywords: *intelligence metrics, artificial life, stigmergy, distributed intelligence*

1. INTRODUCTION

"Self-centered – someone who does not think about me." - Coluche (French Comedian)

Egotistically, most people consider another person intelligent when the other person agrees with them. The Turing test is an egregious example of this tendency. A system is intelligent, when its behavior resembles human behavior. While flattering, this measure is not very objective. Objectively, intelligence is a combination of many attributes. These include the ability to:

- Achieve goals
- Compete with others
- Cooperate with others
- Develop new unexpected behaviors
- Adapt to a changing environment

These attributes are necessary but not sufficient for describing intelligence. A truly intelligent system should also interact with its environment, modifying the environment to its advantage. This ability is based on what is called *stigmergy* by Grassé [16].

Grassé coined the term stigmergy while studying highly evolved societies of cooperating individuals. The societies shared the following characteristics:

- Construction of climate controlled communal housing
- Individuals altruistically sacrifice themselves for the common good
- Equitable distribution of work among their members
- Division of tasks among castes of specialized workers
- Domestication of other species
- Creation of logistic networks to support cities and war campaigns

These societies belong to the most universally successful species on earth, controlling most of the air and ground space. They are distinguished by having six legs.

A collective view of intelligence is not limited to the behaviors of insect societies. Cellular Automata research shows how networks of extremely simple automata collectively emulate general computation engines, such as Turing and von Neumann machines [27]. Connectionist methods in artificial intelligence create complex behaviors in a network of extremely simple computation engines [10]. Minsky's *Society of Mind* describes human behavior emerging from interactions among multiple simpler individual entities [19].

Bonabeau's work [3, 4, 5] provides a starting point for an objective definition and measure of intelligence. An individual, or society of individuals, is intelligent when it exhibits a significant degree of stigmergy. It not only adapts to its environment, it interacts with the environment, forcing the environment to adapt to its needs and goals. This interaction is not purely deterministic but results in new behaviors that advance the system towards its goal.

The rest of this paper is organized as follows: Section 2 discusses relevant aspects of cellular automata, on appropriate formalism for studying interactions of distributed systems. Relevant studies of insect colonies are provided in section 3, along with the original concept of stigmergy. Section 4 discusses applications of pheromones and stigmergy to synthetic systems. Some applications reproduce lifelike behaviors. Other applications create synthetic environments using stigmergy-like control mechanisms. Section 5 describes

a distributed system where simple behaviors of local systems combine to produce complex adaptive behavior for the network. A possible stigmergy scale is further discussed in section 6, which concludes the paper.

2. CELLULAR AUTOMATA

A cellular automata (CA) is a synchronously interacting set of elements (network nodes) defined as a synchronous network of abstract machines [1]. A CA is defined by:

- *d* the dimension of the automata
- *r* the radius of an element of the automata
- *d* the transition rule of the automata
- *s* the set of states of an element of the automata

An element's (node's) behavior is a function of its internal state and those of neighboring nodes as defined by *d*. The simplest instance of a CA is uniform has a dimension of 1, a radius of 1, and a binary set of states. In this simplest case for each individual cell there are a total of 2^3 possible configurations of a node's neighborhood at any time step. Each configuration can be expressed as an integer *v*:

$$v = \sum_{i=1}^{1} j_i * 2^{i+1} \quad (1)$$

where: *i* is the relative position of the cell in the neighborhood (left=-1,current position =0, right=1), and j_i is the binary value of the state of cell *i*. Each transition rule can therefore be expressed as a single integer *r*:

$$r = \sum_{v=1}^{8} j_v * 2^v \quad (2)$$

where j_v is the binary state value for the cell at the next time step if the current configuration is *v*. This is the most widely studied type of CA. It is a very simple many-to-one mapping for each individual cell. The aggregated behaviors can be quite complex [11]. Wolfram [27] has created four qualitative complexity classes of CA's:

- *Stable* - Evolving into a homogeneous state.
- *Repetitive* - Evolving into a set of stable or periodic structures.
- *Chaotic* - Evolving into a chaotic pattern.
- *Interesting* - Evolving into complex localized structures.

Two further results show the computational abilities of the CA. Simple CA's can be constructed that reproduce themselves. This was one of the initial concepts von Neumann had in mind when he originated the CA model [11]. CA networks of sufficient size are capable of simulating general computations [17]. Networks containing interactions of extremely simple automata are therefore capable of producing arbitrarily complex aggregated system behavior.

This is related to the ability of neural networks to produce complex behaviors through network interactions among simple threshold devices. Feed-forward and competition networks can infer complex piecewise linear classification functions from a set of examples [10, 22]. These abstractions support the concept that intelligence is a property of aggregated system interactions, rather than individual components. It is worth noting that most connectionist approaches rely on randomly choosing initial conditions in the network.

3. INSECT BEHAVIORS

Artificial life researchers seek new approaches to intelligence, coordination, and self-organization among distributed autonomous systems in insect colony behaviors [24, 21]. Self-organization is very important in living systems. The basic ingredients of self-organization are [3]:

- Positive feedback - includes recruitment and reinforcement of behaviors.
- Negative feedback - counterbalances positive feedback to stabilize the system.
- Amplification of fluctuations - randomness and fluctuations are crucial to system adaptation.
- Multiple interactions - simple behaviors at the micro level aggregate into intelligent adaptations at the macro level.

In addition, arthropods have a number of broadcast signals such as alarms [25]. These primitives are biologically inspired and the basis of many complex animal behaviors such as swarming, flocking, etc.

Insect colonies use pheromones to provide positive and negative feedback signals with these characteristics [14]. Pheromones are natural chemicals secreted by individual animals, and received by other individuals using the sense of smell. They influence the behavior and development of the receivers. Pheromone interactions have been used to model food collection, nest building, task allocation, and war in insect societies [3]. Computer simulations based on these explanations have produced colony behaviors similar to those found in nature [5].

Stigmergy is indirect communication between one or more agents through the environment using pheromone interactions [16]. An individual interacts with its environment depositing pheromones. The specific pheromone left depends on the task being performed by the individual. Pheromones degrade and diffuse over time. They also aggregate as shown in figure 1. In this way, multiple interactions can be combined automatically to provide a single information source describing the aggregate state of the environment. Stigmergy expresses the synergy that occurs when multiple agents form a feedback loop with their environment.

The presence of pheromones in the environment provides dynamic information that regulates individual behaviors. Individual actions aggregate into macro-behaviors and pheromone signals aggregate into macro-information. In this way an agent modifies its environment, and the environment adapts to the needs of the agent.

The best-known example of this is foraging for food by ant colonies. Dorigo has expanded the basic concept into a general optimization methodology [12]. Each ant in a colony

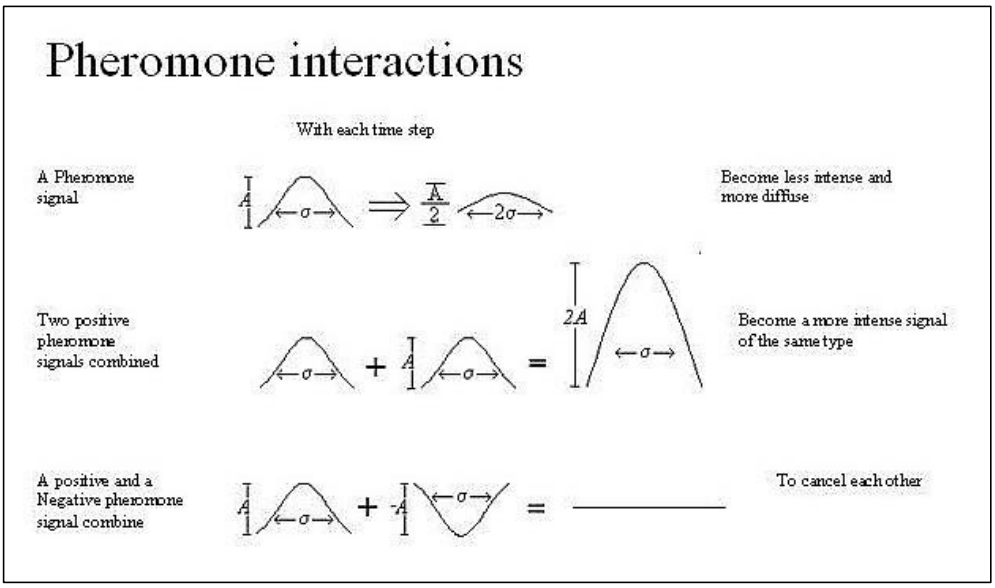


Figure 1 – Pheromone primitive characteristics for information diffusion and reinforcement

performs two basic behaviors, regulated by two basic pheromones:

- *Look for food* – wander in a stochastic manner depositing a "searching for food" pheromone. If the "found food" pheromone is detected, the stochastic movement is weighted to favor movement towards the "found food" pheromone.
- *Bring food to the nest* – when food is located, the ant picks it up. As long as the ant carries food it deposits the "found food" pheromone. It moves in a stochastic manner weighted towards the direction with the strongest "searching for food" pheromone signal.

In [12] and [5] this behavior is analyzed in detail. By heading towards the strongest concentrations of the pheromones, ants tend to follow the direct path. By allowing stochastic deviations, premature convergence to sub-optimal solutions is averted. By aggregating behaviors of many individuals, the system achieves a large degree of robustness.

Note that random decisions play a large role in this behavior. This resembles the use of random initial conditions in neural networks. Many other meta-heuristic approaches, such as genetic algorithms and simulated annealing [7], rely on stochastic, non-deterministic choices to find good quality results.

4. SYNTHETIC ECOSYSTEMS

Self-organizing systems are characterized in Bonabeau [3] by:

- Creation of spatio-temporal structures in initially homogeneous media.
- Co-existence of many possible and reasonable solutions.
- The existence of bifurcations; common in non-linear systems [2].

Self-organizing systems of this type have several appealing aspects, such as robustness and conservation of resources. The existence of multiple possible solutions means that if one solution becomes untenable another can be found. Basing behavior on local decisions using purely local information reduces latency and bandwidth consumption. For these reasons a number of artificial systems have been designed using these principles.

Synthetic stigmergy has been applied to distributed route planning [5], military command and control [23], factory workflow design [9], and telecommunications networks [4]. It provides a convenient formalism for expressing dynamic interactions of multiple agents.

All of these approaches construct a synthetic environment for cooperating agents. Agents change the environment, adding information to it in the form of pheromones. Specific attributes of the pheromone such as speed of dissipation, diffusion rate, and meaning are specific to the individual application. Multiple simple agents then use the information aggregated by the environment to steer their partially stochastic behaviors. Note the similarity between this approach and the CA formalism. Macroscopic interactions between simple individual components provide complex adaptive behaviors.

5. AUTONOMOUS SENSOR NETWORKS

One application of this approach is in sensor networks. Distributed sensor networks use multiple autonomous sensor nodes to provide a sensing system with greater precision and dependability than any component sensor nodes [7]. When multiple sensor nodes survey the same region, redundancy reduces system sensitivity to single points of failure. At any point in time, a single sensor provides a single data point.

Collaborative signal processing aggregates data points into a more reliable global estimate with dependability estimates. This is similar to using multiple experiments to statistically determine a parameter value and its variance [20]. A number of military and commercial applications exist for this technology [6, 13].

Sensor nodes do local processing and relay information among themselves. They self-organize into a coherent whole, forming an ad hoc multi-hop network. Data is relayed from one node to another. Routing choices can be made dynamically using self-organization primitives such as pheromones. Master nodes determine a frequency-hopping schedule that slave nodes follow. Data can be forwarded from one cluster of nodes to the next, until a gateway to the Internet is reached, at which point, a number of user workstations can access the information simultaneously.

Sensor networks have a number of unique aspects. Manual deployment and placement of a large network of sensors would be time consuming and expensive. Ideally the nodes could be deployed automatically. When the number of nodes increases beyond a trivial number, manual network organization becomes problematic as well. Figure 2 illustrates many of the factors that influence network organization and deployment.

When there are a large number of nodes, manual task distribution becomes onerous and time consuming. If conflicts exist in the needs of different user communities the process becomes even more challenging. All of these reasons point to the fact that the networks must be capable of self-organization and autonomous tasking.

Nodes have a finite lifetime, which is shortened by computation, sensing, data transmission, and data reception because they are battery-powered. Most distributed dependability theories are irrelevant to these networks [15]. Distributed dependability verifies the properties of *safety* (lack of undesirable events in the network) and *liveness* (a networks eventual return to a long-term steady state). Since batteries will eventually be exhausted, the network will eventually fail. The property of liveness is impossible to attain. Instead, the system must strive for adaptability. It should reconfigure and tolerate multiple faults. Routing algorithms should avoid creating "hot-spots" that frequently relay data through the multi-hop network, since they will fail much more quickly than the rest of the network. Traffic to relay system housekeeping information should be kept to an absolute minimum.

Traffic patterns for sensor networks differ from those in more traditional *ad hoc* mobile communications networks, such as cell-phones. In traditional *ad hoc* networks, communications are desired between two specific nodes (customers). The network routing protocol needs to find the node no matter where it is located in the network. Sensor networks have the opposite task. Information is required about a specific location. The node identity is irrelevant. For this reason, routing is data-centric.

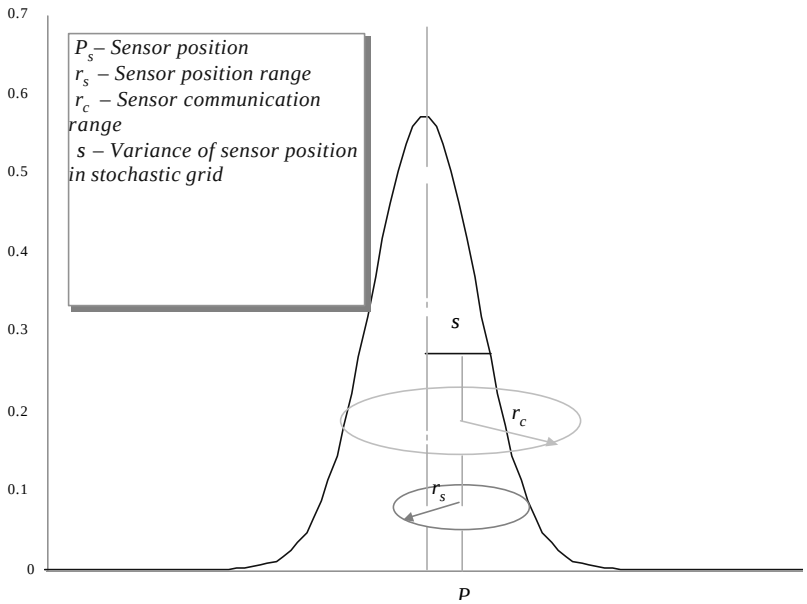


Figure 2. When networks are manually or autonomously deployed and configured, a number of factors need to be considered. These include sensor range r_s and communications range r_c. In the current implementation $r_s > r_c$. In addition to this the nodes position is generally known from GPS units and have an associated uncertainty

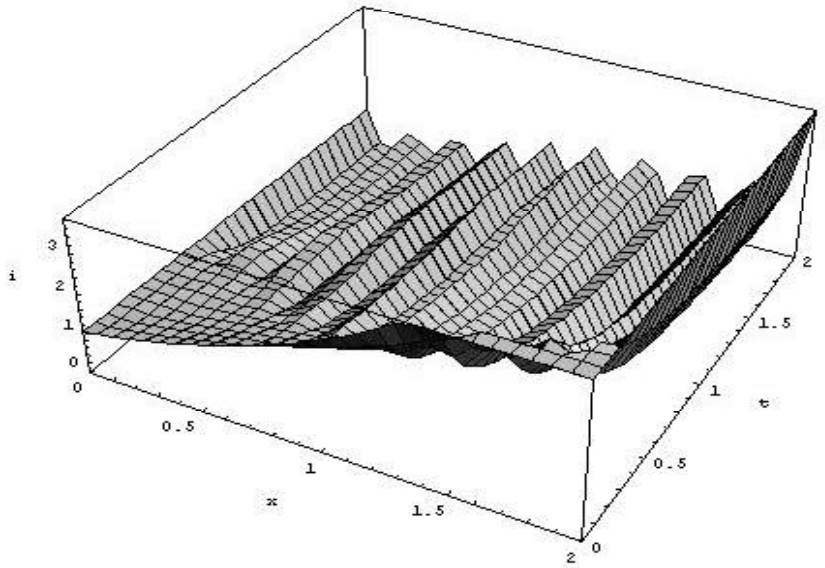


Figure 3 An example transient effect modeled for packet density in a simple sensor network.

In cell-phone networks, communications occur between nodes in the network for a conversation of unknown length. Conversation length is often modeled as an exponential distribution. A reasonable goal is to find a path through the network, which asymptotically approaches the least cost path over time. In sensor networks, queries tend to be punctual. They are either to inform the user of the current state, or inform the user quickly when a given event occurs. It will not be unusual for a single packet to be sufficient. Asymptotic optimality is irrelevant. Transient effects that can be ignored in other systems become much more important. Figure 3 illustrates an example of the transient effects modeled for a simple network.

The Reactive Sensor Network project at the Pennsylvania State University Applied Research Laboratory implements a mobile code infrastructure that augments sensor network adaptability [8]. This approach is inspired by the *active network* paradigm [26].

This approach helps in implementing a self-tasking network. Specific node work assignments need not be known in advance. The software can be reconfigured and modified as needs arise. Similarly if the battery fails on a node performing an essential task, another node can download the software needed to replace it. Figure 4 provides a view of how the individual nodes interact to form a single multicomputer. Notice that it is a macroscopic multicomputer aggregating the behaviors of its autonomous components.

Pheromone based control is also possible. One candidate pheromone is remaining battery power. Another candidate pheromone is distance to an Internet gateway. Combining the

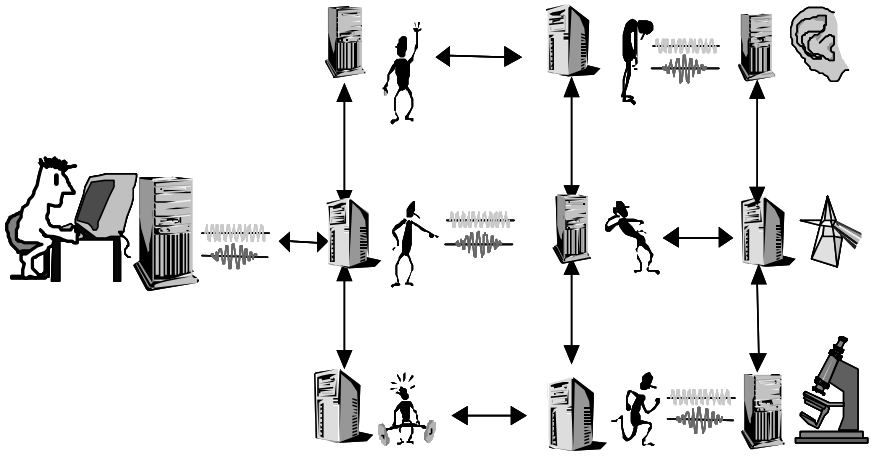


Figure 4. The network is a large computing system formed of individual nodes and sensing devices. Task distribution is determined based on current workloads.

two pheromones provides a self-organizing sensor data network synthetic eco-system that avoids the creation of "hot-spots." This extends the useful lifetime of the network. The CA formalism is useful in exploring a distributed system like autonomous sensor networks (figure 4). Behaviors of individual nodes can be simple and guided by local information. What is important is that the entire system develops complex adaptive behavior from interactions among the nodes.

6. CONCLUSION

Unfortunately, most intelligence metrics are inherently subjective. They often translate into the Supreme Court's metric for pornography: "I know it when I see it." Equally unfortunately, most of us recognize intelligence mainly when looking in the mirror. An example of this type of subjective and narcissistic metric is the Turing test. For the concept of intelligence to be useful, it needs an objective metric.

Can a purely deterministic system be considered intelligent? If this is the case, arithmetic equations and statements of fact are legitimate candidates for intelligence. To the contrary, intelligence is beyond rote memorization and execution of explicit recipes. Intelligence has a creative aspect. An intelligent entity must provide unexpected, creative, appropriate, results. This implies a nondeterministic, random, or stochastic component. Distributed networks of simple interacting automata are robust examples and are capable of performing general computations [27].

Two existing qualitative hierarchies provide objective metrics of intelligence:

- **Chomsky's language hierarchy:** (1) regular grammars recognized by finite state automata, (2) context free grammars recognized by push-down automata, (3) context sensitive languages recognized by linear bounded automata, and (4) recursively enumerable sets recognized by Turing machines [18].
- **Wolfram's complexity classes of CA's:** (1) stable, (2) repetitive, (3) chaotic, and (4) interesting.

To measure the IQ of intelligent systems another qualitative scale is needed that measures systems interactions with their environment:

- **Nonadaptive** – most systems
- **Adaptive** – can regulate parameters to fit environmental conditions. Most controllers would be in this class.
- **Self-Organizing** – adapt to their environment and autonomously reorganize as required. [12] and [8] are examples of this class.
- **Full stigmergy** – modify the environment to suit their goals. Nest building termites, wasps, and humans are in this category.

Discussing intelligent systems presupposes that intelligence is not a purely human attribute. It is an attribute in both living and artificial systems. For that reason, it is appropriate to use concepts from biological studies of non-human intelligence. In simulations of insect societies and construction of artificial systems, stigmergy has been the key to designing robust, creative, emergent behaviors. An appropriate metric for comparing intelligence should be based on the system's stigmergy, stigmergy being the system's ability to interact with and modify its environment to advance the system's goals.

7. ACKNOWLEDGMENTS

This effort is sponsored by the Defense Advance Research Projects Agency (DARPA) and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0520 (Reactive Sensor Network). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author's and should not be interpreted as necessarily represent the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

8. REFERENCES

[1] Adami, C., *Introduction to Artificial Life*, Springer Verlag, New York, 1998.

[2] Alligood, K.T., Sauer, T.D. and Yorke. J.A., *Chaos: an introduction to dynamical systems.* Springer Verlag, New York. 1997.

[3] Bonabeau, E., Theraulaz, G., Deneubourg, J-L, Aron, S. and Camazine, S. "Self-organization in social insects," *Working Papers of the Satna Fe Institute 1997*. http://www.santafe.edu/sfi/publications/Working-Papers/97-04-032.txt, 1997.

[4] Bonabeau, E., Henaux, F., Guerin, S., Snyers, D., Kuntz, P., and Theraulaz, G., "Routing in Telecommunications Networks with 'Smart' Ant-Like Agents," *Working Papers of the Satna Fe Institute 1998*. http://www.santafe.edu/sfi/publications/Working-Papers/98-01-003.ps. 1998.

[5] Bonabeau, E., Dorigo, M., and Theraulaz, G., *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York 1999.

[6] Bonnet, P., Gehrke, J., Mayr, T., and Seshadri. P, Query Processing in a Device Database System. Ncstrl.cornell/TR99-1775. http://www. ncstrl.org, 1999.

[7] Brooks, R. R. and Iyengar, S., *Multi-Sensor Fusion: Fundamentals and Applications with Software*. Prentice Hall PTR, Upper Saddle River, NJ. 1998.

[8] Brooks, R.R., et al. "Reactive Sensor Networks: Mobile Code Support for Autonomous Sensor Networks," *Distributed Autonomous Robotic Systems DARS 2000*. Accepted for Publication. Springer Verlag. October 2000.

[9] Brueckner, S.A., *Return from the Ant.* Ph. D. dissertation. Humboldt University, Berlin. 2000.

[10] Davalo, E. and Naim, P., *Des Reseaux de Neurones*. Editions Eyrolles, Paris, 1989.

[11] Delorme, M., "An introduction to cellular automata," *Cellular Automata: a Parallel Model*. M. Delorme and J. Mazoyer (eds). pp. 5-50. Kluwer Academic Publishers, Dordrecht. 1999.

[12] Dorigo, M., Manniezzo, V. and Colorni, A., "The Ant System: Optimization by a colony of cooperating agents," *IEEE Transactions on SMC-Part B*, vol. 26, no. 1, pp. 1-13., 1996.

[13] Estrin, D., et al. Next Century Challenges: Scalable Coordination in Sensor Networks. ACM MobiCom 99, Seattle, WA, 1999.

[14] Free, J., *Pheromones of Social Bees*, Chapman and Hall, London, 1987.

[15] Gaertner, J., Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments. ACM Computing Surveys, Vol. 31, no. 1, 1-26, 1999.

[16] Grassé, P.P., "La reconstuction du nid et les coordinations inter-individuelles chez Bellicositermes Natalensis et Cubitermes sp. La theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs," *Insect Sociology*, vol. 6, pp. 41-84, 1959.

[17] H. Gutowitz, *Cellular Automata: Theory and Experiment*, MIT Press, Cambridge, MA, 1991.

[18] Hopcroft, J.E., and Ullman, J.D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA. 1979.

[19] Minsky, M., *The Society of Mind.* Simon and Schuster, New York. 1986.

[20] Neter, J.,Wasserman, W. and Kutner, M.H., *Applied Linear Regression Models*, Irwin, Burr Ridge, IL, 1989.

[21] Langton, C.G., ed. *Artificial Life: An Overview.* MIT Press. Cambridge, MA. 1996.

[22] Pandya, A.S. and Macy, R.B., *Pattern recognition with Neural Networks in C++*. CRC Press, Boca Raton, FL. 1996.

[23] Parunak, H. and Brueckner, S.,"Synthetic Pheromones for Distributed Motion Control," *JFACC Symposium in Advances Enterprise Control*. November 1999.

[24] Resnick, M., "Learning about Life," *Artificial Life*. Vol. 1, no 1-2, Spring 1994.

[25] Skirkevicius, A., "Some Characteristics of Insect Pheromone Communication," *Sensory Systems and Communications in Arthropods*. Pp. 55-61. Birkhaeuser Verlag, Basel. 1990.

[26] Tennenhouse, D. L,. J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and Minden, G.J., A Survey of Active Network Research. *IEEE Communications Magazine*, vol. 35, no. 1, 80-86, 1997.

[27] Wolfram, S., *Cellular Automata and Complexity*. Addison-Wesley. Reading, MA. 1994.

Performance Self-Assessment by and for Regulation in Autonomous Agents

Elpida S. Tzafestas
Institute of Communication and Computer Systems
Electrical and Computer Engineering Department
National Technical University of Athens
Zographou Campus, Athens 15773, GREECE.
brensham@softlab.ece.ntua.gr

ABSTRACT

We are studying the problem of connecting intelligence to performance in the context of autonomous agents with very limited capabilities, where performance is suspected to include very few parameters and be easier to quantify than in more complex cases. We are reviewing and comparing three behavioral models that solve three typical autonomous agent problems, the explorer robot [1][2], the food-collector ant [3] and the cooperative tit-for-tat agent [4]. In all three cases and despite the apparent differences between them, we have defined a single problem-dependent performance measure and, on that basis, we have found that the most intelligent among several alternative models, i.e., the one that to the eyes of an observer achieves better performance, is a self-regulatory model involving a two-level regulatory process and an internal variable representing the state of the problem-solving process, thus self-assessing recent performance. The power of the agent lies in the second level and regulation mechanism, that is problem-dependent, and that has been shown to achieve the highest performance among many alternative models in all three problems. The whole design thus allows the agent to assess its actual performance and correct its behavior by modifying accordingly the first-level regulation rates, or equivalently by adapting the first level regulation law. From a symmetrical point of view, the agent may also be thought of as predicting the future state of the environment and adapting accordingly. The self-regulatory process appears therefore as both the means to effective performance assessment and the low-level prerequisite to enhanced intelligence.

1. INTRODUCTION : FUNDAMENTAL CONCEPTS

We are studying the problem of connecting intelligence to performance in the context of autonomous agents with very limited capabilities, where performance is suspected to include very few parameters and be easier to quantify than in more complex cases. By definition, the bottom-up study of intelligence relies on two axioms, evolution and interaction. The axiom of evolution states that higher forms of intelligence appear as a result of an evolutionary process that proceeds from simpler to more complex forms. Complex intelligence thus containts and requires antecedent simpler intelligence. On the other hand, intelligence has no absolute value, but depends on and is the result of dynamic interaction with a changing world. From an evaluation point of view, intelligence is not a well-defined nor a well-specified property, but it depends on an observer's point of view, or as Brooks [5] says "intelligence is in the eye of the observer". An agent demonstrating intelligence through dynamic interaction with a changing world has to be responsive to its environment and adaptive to a range of unpredicted events and situations. For the sake of enhanced stability, adaptivity methods should better be constructed or "controlled" by the agent itself. On the other hand, we, as designers of autonomous agents, are seeking universal design laws that will make our job easier in the long term. To this end, we are investigating a number of classical autonomous agents problems in an attempt to identify common design solutions, that is design solutions that share design principles.

In what follows, we are reviewing and comparing three behavioral models that solve three typical autonomous agent problems, the explorer robot [1][2], the food-collector ant [3] and the cooperative tit-for-tat agent [4]. In all three cases and despite the apparent differences between them, we have defined a single problem-dependent performance measure and, on that basis, we have found that the most intelligent among several alternative models, i.e., the one that to the eyes of an observer achieves better performance, is a self-regulatory model involving two regulatory processes and an internal variable representing the state of the problem-solving process, thus self-assessing recent performance.

The crucial internal agent variable has to be regulated within bounds. The goal of the agent is to either bring it to a limit (say 0) or prevent it from reaching the extremes. This design step depends on the definition of a quantitative environment or problem state, that will be next used as a metric to evaluate different design alternatives.

Regulation occurs using positive feedback, so that the agent's variable follows the tendency of the external world that it

tries to represent, although the value of the variable almost never coincides with the truth, but rather it maintains a representational distance from it. At a second or meta level, another regulation process takes place that regulates the rates of the first level using negative feedback. The power of the agent lies precisely in the second level and regulation mechanism, that is problem-dependent, and that has been shown to achieve the highest performance among many alternative models (including the one without meta-regulation) in all three problems.

The whole design thus allows the agent to assess its actual performance and correct its behavior by modifying accordingly the first-level regulation rates, or equivalently by adapting the first level regulation law. From a symmetrical point of view, the agent may also be thought of as predicting the future state of the environment and adapting accordingly, because conceptually the negative regulation law is the following:

```
If (the world diverges from the agent's
   representation of it)
then (in the future) adapt so as to get
   closer to the world,
else (in the future) adapt so as to amplify
   differences from the world.
```

As a conclusion, the three case studies show that when an autonomous agent problem may be formulated as a regulation problem, the most intelligent alternative model, i.e. the one achieving highest performance, is one that continuously assesses its own performance and regulates its internal parameters accordingly. Therefore, in these cases intelligence appears as the result of low level self-evaluation and regulation.

2. CASE STUDY I : EXPLORER AGENTS

2.1. *The Problem*

A typical problem encountered in the behavior-based robotics literature is that of ***exploration*** : a set of agents (robots) lands on a planet with the mission to explore its surface for samples of minerals having certain properties. The robots arrive in a spaceship that serves as the planetary base in the course of the mission. The mission is accomplished when the whole surface contained within a certain distance from the base is explored, i.e., when the agents have "swept" the whole area and exhausted the sources of interest (cf. for instance [6][7]). The agents are supposed to return to their base once their mission is accomplished.

The exploration problem has been traditionally tackled from a "functional" point of view : "How does one or more agents sweep a delimited area to exhaust the sources of interest ?". The answer to this question is a control system, an architecture, that allows an agent to navigate, perceive, detect minerals etc., in order to sweep the area in question. A solution such as those encountered in the literature (for instance [8]) that comprises a random component and even without spatial reasoning or learning, statistically ensures the coverage of the interest field and the exhaustion of the mineral sources.

However, from a more "cognitive" point of view, this functionality alone does not respond to the crucial question : "***How do the agents know that they have swept the whole area, or that they have accomplished their mission ?***". In order to answer to that question, we have to reformulate the description of the sweeping task, in a way so as to include an expression, analytical or other, that represents the termination criterion, that is the exhaustion of the mineral sources. To this end, it is sufficient to define an environmental variable, the density of mineral sources, which characterizes the state of the explored area at any moment. In what follows, this density will be denoted as p_w. The explorer-sweeper agent's goal becomes therefore to bring the value of that variable to 0. We will see that an agent having a representation of that variable constitutes a simple solution to this description problem.

Lastly, we seek an agent model that would "optimize" performance, i.e. that would allow an agent to accomplish its mission as fast as possible.

In our simulations, the world under exploration is defined as a square around the central base : the size of the world is therefore the length of the square's edge (the results reported have been obtained in a 25x25 world). The agent's basic control system, as well as the simulation details, is given in [1][2]. We are analyzing next the single agent case, whereas the multiple agents case is studied in [1][2].

2.2. *The Solution : Reformulation of the Problem*

We come now to the second question : "How does the agent know it has swept the whole area in order to return to the base ?". It needs a way to detect the degree of task completion or else a termination criterion (sweeping completed). The only parameter of the task that can be useful to the development of a termination criterion is the source density in the world $p_w(t)$. If the agent knew in advance its initial value $p_w(0)$, we could define as termination criterion a formula such as *{$p_w(0)$ * sqr(r) samples have been collected}* (where *r* is the size of the square's edge, here 25). However, this criterion is not robust because if a sample is not detected, the agent will never terminate (on the other hand, we could certainly allow ourselves to miss a couple of samples).

A simple solution to this problem is to estimate continuously the value of $p_w(t)$ and, given that it falls to 0 as a side effect of the agent's activity, take as a termination criterion $p_w(0)=0$. Estimation of the value of $p_w(t)$ involves then a representational variable which is local to the agent ($p_a(t)$) and

may be done through a simple formula of proportional adaptation :

Representational variable : $p_a(t)$
Proportional adaptation (window w, rate r) :
$p_a(t) = p_a(t-w) + diff * r$
$diff = p_{comp} - p_a(t-w)$
p_{comp} = number of picked samples / number of moves (during the adaptation window)
Termination criterion :
$p_a(t) < e_p$
where e_p is a small threshold (here, e_p=0.001)

The p_{comp} is the agent's estimate of p_w as computed during the adaptation window and the proportional law ensures that the estimate's update does not take place too quickly. This representation/adaptation system shows the advantage of robustness in front of perturbations/manipulations such as reinitialization of $p_w(t)$ during sweeping. Figure 1 illustrates the coevolution of the two variables $p_w(t)$ and $p_a(t)$. As is shown in the figure, ***the representational variable allows the agent to always solve its termination problem without ever taking the real value of the variable it represents*** (except a crossing point). Both variables fall progressively to 0 without ever taking the same value — we could say that $p_a(t)$ "follows" $p_w(t)$. Actually, the rapid rise of $p_a(t)$ in the beginning of the sweeping phase is due to the presence of a sensor of distant samples that makes the agent head toward the mineral sources minimizing its erratic behavior in a way that most of the visited places contain samples. The value of $p_a(t)$ falls then because the value of $p_w(t)$ decreases as a side-effect of the agent's activity who finds less and less samples.

2.3. *On Efficiency : Meta-Regulation*

Next, we proceeded to study the relation between the adaptation system's w and r parameters and the initial world value $p_w(0)$. The system has been simulated for several values of w and r in several initial world densities. The simulation results for three sets of adaptation parameters (quick, medium or slow adaptation) in a medium initial world density are given in fig. 1.

The quick adaptation is more operational than the medium one, which is in turn more operational than the slow one (always according to the task duration criterion). However, the quicker the adaptation, more fluctuations it shows, and the slower the adaptation, more delays it shows. Furthermore, the same parameter setting gives different results in different world densities : the difference in the results is reflected on the shape of the curves (for more curves, refer to [1][2]). More particularly, the agent's response to different perturbations (the shape of the curve of $p_a(t)$) differs according to the boundary condition ($p_w(0)$) : for the same parameter setting, the agent finishes its task more or less quickly according to the value of $p_w(0)$, that is the duration of the interval between the moment of picking of the last sample and the definitive return of the agent to the base is very variable. It seems therefore that to ensure the agent's operationality in different worlds, we need to find a means to combine the operational advantages of quick adaptation with the advantages of slow adaptation as far as curve regularity is concerned.

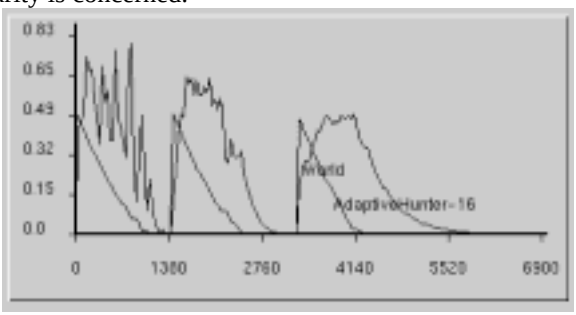


Figure 1. Performance of the agent for different parameter settings in a medium initial world density, $p_w(0)$=0.5 ($p_a(0)$=0.15), t_1=1437, t_2=3278, t_3=6821.
First part (quick adaptation) : w=15, r=0.3, dt_1=1437.
Second part (medium adaptation) : w=30, r=0.2, dt_2=1841.
Third part (slow adaptation) : w=45, r=0.1, dt_3=3543.

More precisely, we need a quick adaptation near the end (to terminate quickly), but a slow adaptation during picking (to avoid fluctuations). We have then to find a way to stabilize to the right parameter setting *on-line*. Otherwise stated, ***we need a meta-adaptation system***.

Meta-adaptation has to affect the w and r parameters in a way that adaptation becomes quicker when p_{est} is sufficiently close to $p_a(t)$ and slower when it is far from it. This meta-adaptation law translates the fact that the world is more reliable when it is not much different from the agent's idea about it, otherwise it should not be taken too seriously.

Meta-adaptation :
If $|diff|$ (= $|p_{comp}-p_a(t-w)|$) $\leq f_p$,
then quicker adaptation
$r \rightarrow r_{max}$, $w \rightarrow w_{min}$
$(r = r + r_r * (r_{max}-r),\ w = w + r_w * (w_{min}-w))$
otherwise slower adaptation
$r \rightarrow r_{min}$, $w \rightarrow w_{max}$
$(r = r + r_r * (r_{min}-r),\ w = w + r_w * (w_{max}-w))$

Figure 2 gives the results of applying the meta-adaptation system in three initial world densities ; as is shown in the figure, the agent's response (the shape of the curve) is the same for all three exemplary densities, or else the residue of mission duration after picking the last sample is approximately the same in all three cases.

We have shown in [1][2] that the operationality of the agent with the meta-regulation law does not depend qualitatively on the values of w_{min}, w_{max}, r_{min}, r_{max}, r_w and r_r. Furthermore, the initial condition ($p_a(0)$) plays no role either.

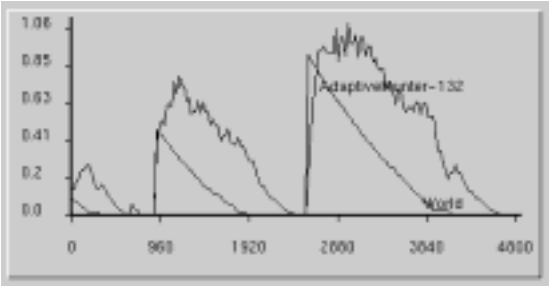


Figure 2. Performance of the agent with a meta-adaptation system for three initial world densities, low ($p_w(0)$=0.1), medium ($p_w(0)$=0.1) and high ($p_w(0)$=0.9) ($p_a(0)$=0.15). dt_1=897, dt_2=1663, dt_3=2211 (t_1=897, t_2=2560, t_3=4771). (f_p=0.1, w_{min}=15, w_{max}=40, r_{min}=0.15, r_{max}=0.3, r_r=r_w=0.2)

3. CASE STUDY II : TRAIL-MAKING ANTS

3.1. *The Problem*

In another variant of the previous problem ([8][9][10]) there are a few large sources distributed in the world. The solution in this case consists in allowing a robot to lay down trails or "crumbs" while carrying a source sample to the home base, that another robot or itself may follow to arrive to the source quickly. A different version of the problem considers that trails laid down by the robots evaporate slowly, in the same way as pheromone quantities laid down by real ants in the physical world ([11]).

The first complete solution has been given in [10], where a number of increasingly complex and increasingly satisfactory solutions have been analyzed. The Tom Thumb robot is able to successfully build, reinforce and correctly use trails from the home base to the source, while the Docker robot [10] uses an additional mechanism of sample "theft" from neighbors, which allows robots to build chains resembling harbor Dockers. The motivation for our work has been our feeling that the Tom Thumb robot as defined is not stable because it assumes unbounded numbers of "crumbs", which is not physically possible, and which would show in a real robotic implementation. A detailed presentation of what follows may be found in [3].

The Tom Thumb robot's behavioral diagram as described in [10] is as follows :

```
If (carrying samples)
    If (back home) lay down samples
    Else {go home, lay down 2 crumbs}
Else
 If (found samples) pick up samples
 Else
    If (crumb or stimulus sensed) (*)
       {follow stimulus, pick up 1 crumb}
    Else move randomly

(*) In the Docker robot, the condition
(crumb or stimulus sensed) is replaced
by (crumb or stimulus or loaded robot
sensed).
```

The Tom Thumb robot lays down two crumbs while homing, and picks up one crumb while following crumbs or stimuli. Unless otherwise stated, all simulations reported below use a 30x30 grid world with a large source at one of the corners and a population of 10 robots starting with 50 crumbs each. Robots may sense a sample or crumb from a distance of up to 3 grid cells.

We have simulated the behavior of the system as is, by measuring the quantities of crumbs deposited in the world or owned by individual agents. As was expected, the quantities of crumbs owned by robots generally fall below zero, while the quantity of crumbs deposited in the world may rise without limit. The exact values of these quantities depend on the problem parameters (distance from source to home base, number of robots and source size) that define the expected number of robot trips source-base necessary to complete the task.

3.2. *The Solution : Reformulation of the Problem*

An apparent question arising at this point is, "what if we just constrain robot behavior so as not to lay down crumbs when it does not have any ? aren't crumbs deposited so far enough ?" We have been able to see in several experiments that, first, depending on the problem parameters, the total quantity of crumbs might not be sufficient, in which case the path to the source will be disconnected, and, second, when it is sufficient — for instance if we start the above experiment with 1000 crumbs per agent — the total number of crumbs deposited in the world may rise tremendously. This last condition generates an important problem : the robots will continue being attracted for a long time to an empty source, that is, the surplus crumbs will be misleading. This observation brings us to the actual formulation of the above trailing problem :

We are seeking a laydown-pickup mechanism such that a trail to a source is built quickly and reinforced while the source exists and vanishes shortly after the source is exhausted.

The problem of agent crumb exhaustion lends itself to a simple solution. Every time a robot needs to lay down or pick up crumbs, it should do it in a way so as to preserve its own quantity of crumbs within some desired bounds $crumbs_{min}$ and $crumbs_{max}$, by using the following laws :

For laydown $crumbs(t+1) = crumbs(t) + r_l * (crumbs_{min} - crumbs(t))$

For pickup $crumbs(t+1) = crumbs(t) + r_p * (crumbs_{max} - crumbs(t))$

This simple regulation mechanism ensures that no agent will ever run out of crumbs completely. However, the absolute (real-

valued) quantity of crumbs deposited or collected at each cycle will depend on the state of the agent : an agent with many crumbs will lay down more and pick up less than an agent with just a few crumbs remaining. This arrangement allows for trails to be built rapidly (because agents in the beginning tend to lay down large quantities of crumbs) and to vanish quickly (because agents toward the end of the task have statistically only a few crumbs, so they tend to pick up large quantities of crumbs). In what follows it will be assumed that $crumbs_{min}$=10 and $crumbs_{max}$ = 100, for all agents.

3.3. *On efficiency : Meta-Regulation*

A large laydown rate will be beneficial in the start and middle of the task, when the agents would like to build and reinforce a trail quickly, while a large pickup rate would be beneficial toward the end of the task, when the agents would like to destroy the trail to the exhausted source as quickly as possible. While a given parameter setting would be more desirable than another one in a particular context, our goal as designers should be to ensure the better behavior *globally*, i.e., to ensure that the system will "discover" or identify the proper parameter setting in each situation.

Consequently, what we really want is *not* a particular parameter setting, but a mechanism that will allow a robot to lay down more and pick up less crumbs at the beginning of the task (so as to build and reinforce the path) and vice versa toward the end (so as to destroy it quickly). To this end, a measure of the state of the task must be available. The only such measure that a robot may have is the number of the crumbs in the world. However, since this quantity cannot be directly perceivable, we have used an estimate of it, simply the number of crumbs at the current position of the robot. This estimate is used as follows :

For laydown
If crumbs(t) >= world_crumbs_estimate
$r_l(t+1) = r_l(t) + r_{rl} * (r_{lmax} - r_l(t))$
else $r_l(t+1) = r_l(t) + r_{rl} * (r_{lmin} - r_l(t))$
For pickup
If crumbs(t) >= world_crumbs_estimate
$r_p(t+1) = r_p(t) + r_{rp} * (r_{pmin} - r_p(t))$
else $r_p(t+1) = r_p(t) + r_{rp} * (r_{pmax} - r_p(t))$

As is obvious from the formulae, the rate of crumb laying increases when the robot owns more crumbs than may be found in its current position and decreases otherwise. Inversely, the rate of crumb picking increases when the robot owns less crumbs than may be found in its current position and decreases otherwise.

Figure 3 gives a typical result of the application of the above model. Surprisingly enough, the self-regulation of the laydown and pickup rates not just does change the qualitative behavior of the agents (the quantity of crumbs in the world rises quickly to a fairly high value, stays close to it during the task, and falls back quickly to zero when the source is exhausted, while showing far less fluctuations than in the previous case), but it improves results quantitatively as well : in all runs, including the one depicted, the duration of the task has been shorter than with the non-regulated model.

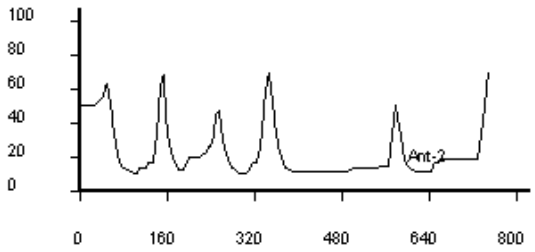


Figure 3. Quantity of crumbs owned by a meta-regulated agent in a typical run. It fluctuats between the upper and lower limits.

4. CASE STUDY III : ADAPTIVE TIT FOR TAT AGENTS

4.1. *The Problem*

A major issue on the intersection of artificial life and theoretical biology is cooperative behavior between selfish agents. The cooperation problem states that each agent has a strong personal incentive to defect, while the joint best behavior would be to cooperate. This problem is traditionally modeled as a special two-party game, the Iterated Prisoner's Dilemma (IPD).

At each cycle of a long interaction process, the agents play the Prisoner's Dilemma. Each of the two may either cooperate (C) or defect (D) and is assigned a payoff defined by the following table.

Agent	Opponent	Payoff
C	C	3 (= Reward)
C	D	0 (= Sucker)
D	C	5 (= Temptation)
D	D	1 (= Punishment)

Usual experiments with IPD strategies are either tournaments or ecological experiments. In tournaments, each strategy plays against all others and scores are summed in the end. In ecological experiments, populations of IPD strategies play in tournaments and successive generations retain the best strategies in proportions analogous to their score sums.

The first notable behavior for the IPD designed and studied by Axelrod [12] is the Tit For Tat behavior (TFT, in short) :

Start by cooperating,
From there on return the opponent's previous move.

This behavior has achieved the highest scores in early tournaments and has been found to be fairly stable in ecological settings.

The best designed behavior found so far in the literature is GRADUAL [13] which manages to achieve the highest scores against virtually all other designed behaviors. This behavior starts by cooperating and then plays Tit For Tat, except that it does not defect just once to an opponent's defection. Instead, it responds by playing blindly (nxD)CC, where n is the opponent's number of past defections. That is, GRADUAL responds with DCC to the first opponent's defection, DDCC to the second, etc. The justification given for the performance of this behavior is that it punishes the opponent more and more, as necessary, and then calms him down with two successive cooperations.

The motivation for our work has been our conviction that a behavior comparable to GRADUAL could be found, that has not permanent, irreversible memory. Instead, we are after a more adaptive tit-for-tat based model that would demonstrate behavioral gradualness and possess the potential for stability in front of changing worlds (opponent replacement etc.).

Before proceeding, let us examine the high scores that GRADUAL obtains against other behaviors. Designed behaviors found in the literature usually fall in one of three categories :

- Behaviors that use feedback from the game, usually cooperative behaviors unless the opponent defects, in which case they use a retaliating policy (tft, grim, gradual, etc.).
- Behaviors that are essentially cooperative and retaliating, but start suspiciously by playing a few times D in the beginning, so as to probe their opponent's behavior and decide on what they have to do next. For example, suspicious tft (STFT) and the "prober" behavior of [13].
- Behaviors that are clearly irrational, because they don't use any feedback from the game. For example, the random behavior and all blind periodic behaviors such as CCD, DDC etc.

A behavior will maximize its score, if it is able to converge to cooperation with all behaviors of the first two categories and converge to defection against behaviors of the third category. Steady defection against periodic behaviors is necessary in order to achieve the highest possible score (see [4], for details).

The GRADUAL behavior fulfills both of the above specifications, because it responds with two consecutive C's after a series of defections, giving the chance to STFT or prober behaviors to revert to cooperation, and converges to ALLD against irrational behaviors. A solution to the permanent memory problem has to demonstrate the same property.

4.2. *The solution : Reformulation of the Problem*

The adaptive behavior that we are seeking should be essentially tit-for-tat. Moreover, it should demonstrate fewer oscillations between C and D. To this end, it should have an estimate of the opponent's behavior, whether cooperative or defecting, and react to it in a tit-for-tat manner. The estimate will be continuously updated throughout the interaction with the opponent. The above may be modeled with the aid of a continuous variable, the world's image, ranging from 0 (total defection) to 1 (total cooperation). Intermediate values will represent degrees of cooperation and defection. The adaptive tit-for-tat model can then be formulated as a simple linear model :

```
Adaptive tit-for-tat
If (opponent played C in the last cycle)
    then
world = world + r*(1-world), r is the
    adaptation rate
else
      world = world + r*(0-world)
If (world >= 0.5) play C, else play D
```

The usual tit-for-tat model corresponds to the case of r=1 (immediate convergence to the opponent's current move). Clearly, the use of fairly small r's will allow more gradual behavior and will tend to be more robust to perturbations.

Now, let us simulate the behavior of the adaptive tit-for-tat agent against all three types of behaviors described earlier.

- For initially cooperative behaviors with feedback and a retaliation policy, the model cooperates steadily and converges quickly to total cooperation.
- For suspicious or prober behaviors, the model plays exactly like tit-for-tat, while the value of the world variable oscillates around the critical value of 0.5 (see figure 4 against suspicious tft).
- For periodic behaviors, the value of the world variable converges quickly to oscillations around the characteristic value of "number_of_C's/number_of_D's" in the opponent's period.

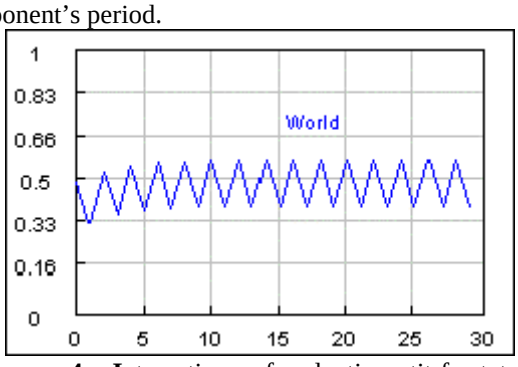


Figure 4. Interaction of adaptive tit-for-tat with suspicious tit-for-tat (r=0.2, world(0)=0.5).

4.3. *On Efficiency : Meta-Regulation*

It can be seen that the previous version of the model suffers from manipulation of the world variable by the opponent. This shows

as stabilization of the agent to an oscillatory behavior (as is the case against stft) or a steady cooperative behavior against irrational agents (as is the case against CCD). To bypass this problem, we exploited our observation that different rates for cooperation and defection (r_c and r_d, respectively) yield different results. More specifically, we observed that the adaptive tit-for-tat agent manages to get opponents such as stft or the prober to cooperate if $r_c > r_d$, while it manages to fall to steady defection against periodic behaviors if $r_c < r_d$.

Thus, what we need at this point is a method for the adaptive tit-for-tat agent to discover whether the opponent uses a retaliating behavior or is just irrational and to adopt accordingly the proper rate setting. We have designed and examined several such variants for estimating the opponent's irrationality and we have finally found the following rule :

Throughout an observation window, record how many times (n) the agent's move has coincided with the opponent's move. At regular intervals (every "window" steps) adapt the rates as follows :
If (n>threshold) then
$r_c = r_{min}$, $r_d = r_{max}$
else $r_c = r_{max}$, $r_d = r_{min}$
The rule may be translated as :
If (the world is cooperative enough) then*
$r_c = r_{min}$, $r_d = r_{max}$
else $r_c = r_{max}$, $r_d = r_{min}$
(*) recall that "my move = opponent's move" is the so-called pavlovian criterion of cooperation ([14])

Note that the agent drops its cooperation rate when the world is assumed cooperative, and increases it otherwise, that is, it uses negative feedback at the rate regulation level.

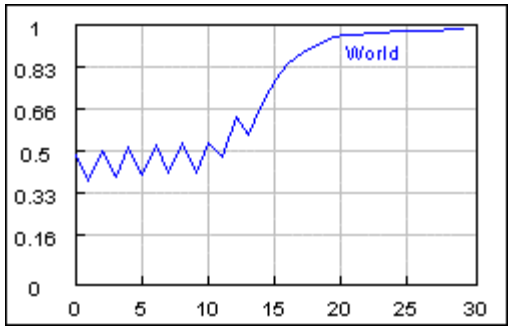


Figure 5. Interaction of the meta-regulated adaptive tit-for-tat agent with suspicious tit-for-tat ($r_c(0)=0.2$, $r_d(0)=0.2$, $r_{max}=0.3$, $r_{min}=0.1$, world(0)=0.5, window=10, threshold=2). Compare with figure 4.

We have shown in simulations that the adaptive tit-for-tat agent with the meta-regulation mechanism converges to the proper behavior against both retaliating and irrational agents. For example, figure 5 gives the behavior of the meta-regulated adaptive tit-for-tat agent against STFT.

Finally, the adaptive agent manages to differentiate between a retaliating agent and an irrational one that has initially the same behavior. The agent first assumes that the opponent is retaliating and becomes increasingly cooperative, but soon finds out that the opponent is actually irrational and reverts to defection.

5. DISCUSSION : ELABORATING THE CONCEPTS

In all three case studies, we have shown that the agent's behavior is based on a critical variable that drives its motivation to act. This variable is coupled with the environment through the agent's behavior. By regulating its own variable, an agent tries to regulate the corresponding world variable. Furthermore, this variable has *cognitive value*, since it represents the agent's idea about the state of the environment. Seen this way, the agent may be thought of as trying to approach or approximate the world variable, i.e., as trying to adapt to its environment. The regulated variables appear to be critical for an agent's survival or operationality, and correspond to what Ashby [15] called *essential variables*.

The operationality of the behavior is ensured through an additional self-regulation mechanism acting on the adaptation rates. This is an important observation, since it is compatible with the dynamical approach to cognition [16], stating that the most important factor in cognitive mechanisms is the nature of dynamics involved. Mechanisms like the ones developed here may be also regarded as a first step toward the realization of autopoietic systems :

> "… an autopoietic system is a homeostat … the critical variable is *the system's own organization* ..." ([17], p. 66)

In sum, we have shown that self-assessment of performance by an agent is done with the aid of a double regulatory process and it allows it to become more operational in its work. This is in line with classical control theory, where regulatory mechanisms are used as the basis of behavior [18]. Inversely, similar regulatory processes may be designed for other problems, provided that the appropriate performance measure (or cognitive variable) and its assessment model are given or may be identified. In this sense, the long term perspective of this work is to build a regulation theory for reactive autonomous agents. To this end, a number of issues have to be investigated :

- How do we identify the critical cognitive variable in each case ? Equivalently, how do we formulate regulation in each case ?
- How many first level rates are necessary ? Equivalently, how many independent regulation processes are there ?

Recall that the explorer agent has one such rate, whereas both the trail-making agent and the adaptive tit-for-tat agent have two of them.

- Which is the meta-regulation criterion ? Note that in all three cases studied this criterion is purely qualitative and problem-dependent. Equivalently, this issue translates to "How can we observe and qualify a regulatory process ?".
- What is the nature of the meta-regulation dynamics ? A few initial experiments show that most probably a "bang-bang" dynamics (high-low value) is enough, because what counts is the relation between two rates rather than their absolute values.
- Finally, what is the role and value of "behavior in the empty" (without perturbation) ? This behavior is purely agent-specific and may differ among different agents, due to different parameter settings, defining thus the individual "character" of an agent. Initial experiments have shown that the behavior in the empty allows some limited prediction to be made.

As a general conclusion, the answers to questions such as the above could teach us a lesson on the power and potential of regulation mechanisms for apparently qualitative problems. They could also deepen our understanding of the scope and limits of such mechanisms amd prompt us to problems of immediately higher complexity, where regulation would not be enough and why this is so.

6. REFERENCES

[1] Tzafestas, E., *Vers une systémique des agents autonomes : Des cellules, des motivations et des perturbations*, Ph.D. diss., Univ. Pierre et Marie Curie, Paris, 1995.

[2] Tzafestas, E., "Regulation Problems in Explorer Agents", submitted.

[3] Tzafestas, E., "Tom Thumb Robots Revisited : Self-Regulation as the Basis of Behavior", *Proceedings Artificial Life VI*, San Diego, CA, June 1998.

[4] Tzafestas, E., "Toward Adaptive Cooperative Behavior", *Proceedings Simulation of Adaptive Behavior 2000*, Paris, France.

[5] Brooks, R.A., "Elephants don't play chess", in P. Maes (ed.), *Designing Autonomous Agents*, MIT Bradford Press, 1991.

[6] Brooks, R.A. and Flynn, A.M., "A robot being", *Robots and Biological Systems : Towards a new Bionics ?* (P. Dario, G. Sandini and P. Aebischer), 1989, pp. 679-701.

[7] Beckers, R., Holland O.E. and Deneubourg, J.-L., "From local actions to global tasks: Stigmergy and collective robotics", *Proceedings Artificial Life IV* (R. Brooks and P. Maes, Eds.), MIT Press, Cambridge, MA, 1994, 181-189.

[8] Mataric, M., "Designing emergent behaviors : From local interactions to collective intelligence", *Proceedings Simulation of Adaptive Behavior 1992*, 432-441.

[9] Steels, L., "Towards a theory of emergent functionality", *Proceedings Simulation of Adaptive Behavior 1990*, 451-461.

[10] Drogoul, A. and Ferber, J., "From Tom Thumb to the Dockers : Some experiments with foraging robots", *Proceedings Simulation of Adaptive Behavior 1992*, 451-459.

[11] Deneubourg, J.-L., Aron, S., Goss, S. and Pasteels, J.M., "The self-organizing exploratory pattern of the Argentine Ant", *Journal of Insect Behavior* 3(1990):159-168.

[12] Axelrod, R., *The evolution of cooperation*. Basic Books, 1984.

[13] Beaufils, B., Delahaye, J.-P., and Mathieu, P., "Our meeting with gradual: A good strategy for the iterated prisoner's dilemma", *Proceedings Artificial Life V*, Nara, Japan, 1996.

[14] Nowak, M., and Sigmund, K., "A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game", *Nature* 364(1993):56-58.

[15] Ashby, W.R., *Design for a brain - The origin of adaptive behaviour*. 2nd revised edition, Chapman & Hall, 1960.

[16] van Gelder, T., and Port, R., "It's about time : An overview of the dynamical approach to cognition", in *Mind as Motion* : *Explorations in the dynamics of cognition*, by T. van Gelder and R. Port, Cambridge, Mass.: MIT Press, 1995.

[17] Maturana, H.R., and Varela, F. (1980). *Autopoiesis and cognition — The realization of the living*, Dordrecht/ Boston, D. Reidel Publishing.

[18] Slotine, J.-J.E. (1994). Stability in adaptation and learning, *From animals to animats 3, Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, MIT Press, pp. 30-34.

On Measuring Intelligence in Multi Agent Systems

Siva Perraju Tolety
GTE Laboratories
40 Sylvan Road,
Waltham MA 02451 USA
toletys@acm.org

Garimella Uma

Abstract

Intelligent behavior means doing the right thing [1]. Due to the bounded rationality of agents it is not always possible to do the right thing. Hence intelligence implies doing the best possible given the resources an agent or a multi agent system has. So, a measure of intelligence should reflect an evaluation of the process by which the agent or the multi agent systems arrive at exhibiting intelligent behavior. In multi agent systems, intelligent behavior is emergent in nature rather than additive. So, measures of intelligence should attempt to estimate the net resultant behavior rather than the individual fine grained reasoning processes of individual agents. Intelligent quotient measures for the human mind attempt to arrive at a single number based on a battery of tests. This number is a reflection of the individual's standing in his or her group. In this paper we attempt to present our ideas about arriving at such measures for multi agent systems.

1. Introduction

Intelligence is expected to allow an agent to do the right thing. The level of intelligence is reflected in the appropriateness of the actions undertaken by an agent in the given circumstances. Measures of intelligence have been used in the human society for a variety of purposes. These uses range from efforts to identify deficiencies in individuals and help them improve in these areas to efforts to rank people according to their capabilities in a given area.

Research in agents and multi agent systems is maturing and systems are being deployed in real world settings. Consequently, users of these systems would like to evaluate the system, understand its advantages and deficiencies and improve upon the same. Also if multiple intelligent systems purport to accomplish identical or similar tasks, the users of these systems will have a natural interest in making a comparison of the different systems. Similar needs in the human society gave birth to different performance measures. The measures are usually referred using the generic term *Intelligence Quotient (IQ).* These varied measures are based on differing views of intelligence. In Section 2, we briefly outline the differing views of human intelligence, and how these views lead to different perspectives of IQ measurements. In Section 3, we outline different multi agent architectures and highlight various aspects of these architectures that can be measured. In Section 4, we outline a possible measure for multi agent architecture developed for problem solving activities in real time domains. Section 5 presents the conclusions.

2. Intelligence and its Measurements in the Human Society

Intelligence is an abstract concept and reflects in some way the competencies and skills an individual possess. Some of the questions about intelligence include [2]:

- Is mental competence a single ability applicable in a variety of settings? or

- Is competence produced by specialized abilities, which a person may or may not possess independently?

If we can resolve this question about intelligence, the next question what are the metrics by which you can measures these competencies. Do these measures reflect in the every day problem solving ability of the individual? The answers to these questions depend to a large extent on the perspective; one subscribes to, about intelligence. Two popular views about intelligence are

- psychometric views and
- Cognitive-psychology view.

The psychometric view of intelligence places emphasis on scores obtained in carefully designed tests to evaluate specific skills. This view gives rise to the popular notion of intelligence quotient. Several versions of intelligent quotients exist. Usually these numbers are arrived by performing factor analysis on scores achieved on tests about different skills. Thus IQ measures reflect the level of what psychologists call *crystallized intelligence (Gc).* Crystallized intelligence is the ability to apply previously acquired problem solving methods to the current problem. Measures of crystallized intelligence correlate strongly with another aspect of intelligence viz. *fluid intelligence (Gf).* Fluid intelligence is the ability to develop techniques for solving problems that are new and unusual from the problem solver's perspective.

The cognitive-psychology view is that thinking is a process of creating mental representation of the current problem, retrieving information that appear relevant and manipulating the representation in order to obtain an answer [2]. This definition encapsulates the concepts of Gc and Gf. Forming a mental representation of the problem is akin to fluid intelligence and extraction of relevant information is similar to crystallized intelligence. A variety of tests based on the cognitive-psychology view of intelligence are also available.

3. Intelligence in Agents and Multi Agent Systems

Agent architectures reflect the underlying problem solving processes. Agents can be broadly classified as either reactive or deliberative. Reactive agents are usually preprogrammed to respond in particular ways to various stimuli from the environment. Agents with deliberative architectures usually use some reasoning process to arrive at a solution. Reflecting upon the classification of intelligence discussed in the earlier section, we can assume that reactive agents depend on crystallized intelligence while deliberative agents depend on fluid intelligence. So, psychometric based measures are appropriate for reactive architectures, while tests that try and evaluate the reasoning processes are appropriate for deliberative agents. Just as IQ tests target different groups of the society, tests in the agent world should also target specific agent sets. For example we might design a test suite that tries to evaluate spidering skills of Internet spider agents.

The power of multi agent systems is in their property of emergent behavior. Apart from the domain knowledge, multi agent systems possess some special features, which make them very attractive. These are

- communication and
- agent interaction

Agent interaction is achieved in a variety of ways. Coordination is the generic agent interaction mechanism that helps agents in a multi agent setting achieve their individual and common goals. Coordination can be achieved either through cooperative or competitive mechanisms. Communication protocols based on theories like speech acts enable agents to exchange small by semantically rich messages in aid of their problem solving. In this perspective of multi agent systems, the communication and

agent interaction aspects of the systems seem to determine the intelligence or performance[1] of the system. Hence we propose that any performance measure for multi agent systems should in some way be able to rank different systems along these two dimensions. So, an IQ measure for multi agent systems is a function of three separate factors. They are

- domain knowledge(DK)
- individual agent reasoning capabilities (ARC)
- communication (COMM) abilities and
- efficacy of agent interaction AI).

MIQ = *f(DK,IARC, COMM, AI)*

Since we are interested in evaluating multi agent settings, we can ignore the factors that can be attributed to individual agents viz. DK and ARC. This implies we are assuming that all agents are equally capable in a given domain. This assumption though not suitable for rigorous measurements, could however be a good starting point. Hence

MIQ = *f(COMM, AI)*

4. Measuring intelligence / performance in TRACE

TRACE (Task and Resource Allocation in a Computational Economy) is a system of multi agent systems designed to operate under time constraints and load variations [3,4]. TRACE approach to problem solving is based on an adaptive organizational policy. The TRACE system is market based multi agent system. Tasks and resources are allocated to different multi agent systems based on their problem solving load and the price they are willing to pay for the resources. We assume that knowledge can be transferred among agents and thus domain knowledge plays no particular role in the evaluation of the multi agent systems.

Intuitively, we know that the efficiency of a player in a market is determined by how efficiently the multi agent system trades its funds for resources to aid in problem solving activities. Different multi agents systems in TRACE can adopt different policies to decide on their problem solving activities.

Now if we attempt to measure the performance of multi agent systems with different policies in the TRACE setting, what are the attributes that can capture the essence of the equations in the previous section? Multi agent systems sign up or commit for tasks and attempt to complete them. In this process they undertake both communication and agent interaction tasks. These tasks are time bound and task completion beyond a deadline is a wasted problem solving activity. In order to achieve maximum returns for the problem solving activity, multi agent systems will drop some tasks (decommit). The lower the number of decommitments the better the performance. The number of decommitments reflects how much a given system has overreached. It in turn reflects on the shortcomings in its communication, negotiation and problem solving abilities. Thus in the case of the TRACE system we feel that a normalized number of decommitments is an accurate measure of the performance of the multi agent systems.

We have implemented a prototype of the TRACE system, in which it is possible to introduce multi agent systems with different problem solving abilities and processes. We intend to formulate a simple problem to be solved by the multi agent systems. We then intend to measure the number of decommitments made and determine if this measure is a reasonably accurate measure of the performance of the multi agent system.

[1] We assume that higher level of intelligence results in better performance. Consequently, a MAS which is better at a given set of tasks than another MAS can be considered to be more intelligent.

5. Conclusions

In this paper we made an attempt at trying to understand the basis of intelligent measures used in the human society. Research in agent systems and multi agent systems led to the development of architectures that in some way try to mimic the problem solving skills in human beings. Thus the science of intelligent quotient measurements can be applied to the domain of intelligent agents and multi agent systems. In market based agent systems, we propose that the number of decommitments made by an agent along with the resources consumed is a measure of its ability. In more cooperative settings different but appropriate measures need to be designed. We conclude that the measures need to be designed by considering a family of agent architectures. For example we feel that a measure designed for a market based agent system will be ill suited for a cooperative multi agent system. We are currently experimenting with a multi agent system TRACE to understand the criteria for measuring its performance.

References

1. S. Russel and E. Wefald, Do the Right Thing, MIT Press, 1991
2. E Hunt, The Role of Intelligence in Modern Society, American Scientist, July- August 1995.
3. S Fatima, G Uma, T S Perraju, An Adaptive Organizational Policy for Multi Agent Systems, ICMAS 2000, Boston USA. July 2000.
4. S Fatima, G Uma, “AASMAN” An Adaptive Organizational Policy for a society of Market based agents, Sadhana, Academy Proceedings of Engineering Sciences, Indian Academy of Sciences, Vol 23, No. 4, pages 377-392.

On the Development of Metrics for Multi-Robot Teams within the ALLIANCE Architecture

Lynne E. Parker
Center for Engineering Science Advanced Research
Computer Science and Mathematics Division
Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831-6355

ABSTRACT

Quantitatively evaluating the effectiveness of software architectures for multi-robot control is a challenging task. Exacerbating the problem is the fact that architectures are typically constructed to address different design goals and application domains. In the absence of benchmarks that capture the variety of issues that arise in multi-robot coordination and cooperation, the system developer can only evaluate an architecture for its own qualities. In this article, we summarize the metrics of evaluation that we utilized in applying our ALLIANCE architecture [17] to eight different application domains for multi-robot team control. We explore the implications of the metrics we have chosen and offer suggestions on future productive lines of research into metrics for multi-robot control architectures.

Keywords: *Multi-robot cooperation, metrics, ALLIANCE*

1 Introduction

Research work in multi-robot systems has progressed significantly in recent years. Issues that have been studied are diverse, and include task planning and control [1, 17, 12]; biological inspirations [6, 7, 13]; motion coordination [27, 2, 4]; localization, mapping, and exploration [22, 21]; explicit and implicit communication [5, 9]; cooperative object transport and manipulation [23, 25]; reconfigurable robotics [28, 24, 26]; and multi-robot learning [11, 12, 10]. Demonstrations have been given of multi-robot teams performing a variety of tasks, such object pushing, foraging, cooperative tracking, traffic control, surveillance, formation-keeping, and so forth.

However, most of this research is very specific and illustrates only one or two basic concepts per project. Comparisons across different methodologies are difficult and quantitative evaluations of various multi-robot control algorithms are scarce. While this is not unexpected for a field as new as cooperative robotics, enough progress has been made that we believe it is time to begin determining how we identify and quantify the fundamental advantages and characteristics of multi-robot systems. The characteristics most often cited for motivating the use of multi-robot teams are as follows:

- increased robustness and fault tolerance through redundancy,
- a potential for decreased mission completion time through parallelism,
- a possibility for decreased individual robot complexity through heterogeneous robot teams, and
- an increased scope of application due to tasks that are inherently distributed.

Other than direct measures of time, these characteristics are hard to quantify, yet vital to enabling the field to make objective comparisons and evaluations of competing architectures. Thus, much research is needed in this area.

2 Background

Measuring the performance of intelligent systems in general, and multi-robot systems in particular, is a much-understudied topic. Some beginning work has been accomplished by Balch [3], who has developed metrics for measuring multi-robot team diversity. However, little research has addressed the general issues of cooperation that provide guidelines for the quantification and selection of the appropriate cooperative team for any given set of mission specifications. Such a characterization would be a significant step towards the commercialization of cooperative systems, as it would facilitate the design of the appropriate cooperative team for a given application. Issues of particular interest in such a characterization include the following:

- Quantifying the overall system capability versus the system complexity,
- Determining the appropriate distribution of capabilities across robot team members for a given application,
- Ascertaining the most appropriate control strategy for a given robot team applied to a given application so as to maximize efficiency, fault tolerance, reliability, and/or flexibility, and

- Determining tradeoffs in control strategies in terms of desirable traits, such as efficiency versus fault tolerance.

Examples of this type of research include [8], which develops measures of effectiveness and system design considerations for the generic area coverage application, and [14], which compares the power of local versus global control laws for a "Keeping Formation" case study. However, much more work remains to be accomplished towards the development of quantitative comparisons of alternative approaches to cooperative team design. An understanding of the factors that influence the relative performances of various approaches to cooperative control will enable not only an evaluation of existing methodologies, but will also aid in the design of new cooperative control approaches.

Since addressing the issue of quantitative measurement and system integration for the entire field of cooperative robotics is extremely challenging, we have begun work in this area by focusing on our experiences with the ALLIANCE architecture. We developed the ALLIANCE architecture [17] to enable fault tolerant action selection in multi-robot teams. The focus was on an approach that operated successfully amidst a variety of uncertainties, such as sensory and effector noise, robot failures, varying team composition, and a dynamic environment. We have implemented ALLIANCE in eight different application domains in the laboratory. This experience is the basis for our beginning work in the development of general metrics and system integration as it applies to the use of ALLIANCE.

3 Brief Overview of ALLIANCE

We developed the ALLIANCE architecture to enable fault tolerant action selection in multi-robot teams. The focus was on an approach that operated successfully amidst a variety of uncertainties, such as sensory and effector noise, robot failures, varying team composition, and a dynamic environment. The ALLIANCE architecture, shown in Figure 1, is a behavior-based, distributed control technique. Unlike typical behavior-based approaches, ALLIANCE delineates several behavior sets that are either active as a group or are hibernating. Each behavior set of a robot corresponds to those levels of competence required to perform some high-level task-achieving function. Because of the alternative goals that may be pursued by the robots, the robots must have some means of selecting the appropriate behavior set to activate. This action selection is controlled through the use of motivational behaviors, each of which controls the activation of one behavior set. Due to conflicting goals, only one behavior set is active at any point in time (implemented via cross-inhibition of behavior sets). However, other lower-level competencies such as collision avoidance may be continually active regardless of the high-level goal the robot is currently pursuing.

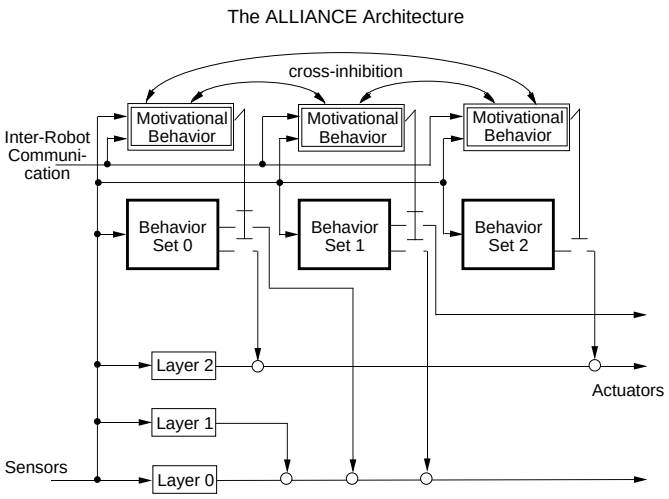


Figure 1: The ALLIANCE architecture for multi-robot cooperation.

The motivational behavior mechanism is based upon the use of two mathematically-modeled motivations within each robot – impatience and acquiescence – to achieve adaptive action selection. Using the current rates of impatience and acquiescence, as well as sensory feedback and knowledge of other team member activities, a motivational behavior computes a level of activation for its corresponding behavior set. Once the level of activation has crossed the threshold, the corresponding behavior set is activated and the robot has selected an action. The motivations of impatience and acquiescence allow robots to take over tasks from other team members (i.e., become impatient) if those team members do not demonstrate their ability – through their effect on the world – to accomplish those tasks. Similarly, they allow a robot to give up its own current task (i.e., acquiesce) if its sensory feedback indicates that adequate progress is not being made to accomplish that task.

We have shown that this approach can guarantee, under certain constraints, that the robot team will accomplish their objectives [15]. We have implemented this approach in a wide variety of applications in the laboratory on several different types of physical and simulated robot systems. Figures 2 and 3 illustrate these different implementations. The implementations include the "mock" hazardous waste cleanup [17], box pushing [20], janitorial service [16], bounding overwatch [16], formation-keeping [14], cooperative manipulation [18], cooperative tracking of multiple moving targets [19], and cooperative production dozing. These implementations and results now give us the basis for studying issues of metrics within this framework.

4 Evaluation of Metrics in ALLIANCE Applications

In [16], the ALLIANCE architecture was demonstrated to have the important qualities of robustness, fault tolerance, reliability, flexibility, adaptivity, and coherence, which we identified as critical design requirements for a cooperative multi-robot team architecture. These broad characteristics, however, were determined based upon qualitative evaluations of the various implementations we have performed. Ideally, we would prefer to have more quantitative metrics of evaluation for these higher-level team characteristics.

On a more application-specific level, we used several metrics to evaluate robot team performance within each of these applications. Table 1 summarizes the metrics we used to analyze the performance of multiple robot teams in eight different ALLIANCE implementations. In these applications, concrete indicators of mission success were used, such as numbers of objects moved, distance traveled, or number of targets within view. Improved mission quality was based upon the time taken to achieve these indicators. This is natural, since a primary benefit of multiple robot teams is using parallelism to achieve mission speedup. In these implementations, no single metric was found to be most useful. The need for a variety of metrics suggests that system performance measures are application-dependent. These examples also illustrate that, for typical applications, the most important issues are *whether* and *how well* the robot team completes its mission.

By focusing on application-specific metrics, however, the broader-perspective qualities of robustness, fault tolerance, adaptivity, etc., are not made explicit. Instead, these characteristics are hidden in the application-specific measures. Thus, any shortcomings in a robot team's ability to operate robustly or with a high degree of fault tolerance, for example, would be measured by an increased time to complete the mission (or by never completing the mission at all), a decreased distance traveled, fewer objects moved, etc. It would be difficult, therefore, to determine the relative levels of contribution of the various broader-perspective qualities (e.g., fault tolerance vs. adaptivity) to changes in the application-specific quantitative measures (e.g., distance traveled). Thus, if one wants to explicitly measure fault tolerance across several control architectures, and/or several application domains, these metrics are not suitable.

An important goal of research in the quantitative evaluation of robot control architectures is, therefore, the development of metrics that enable quantitative measurement higher-level characteristics, including fault tolerance, reliability, flexibility, adaptivity, and coherence. By averaging the results across multiple application domains, we would then be able to explicitly compare alternative control architectures in terms of these important application-independent characteristics. Our continuing research is

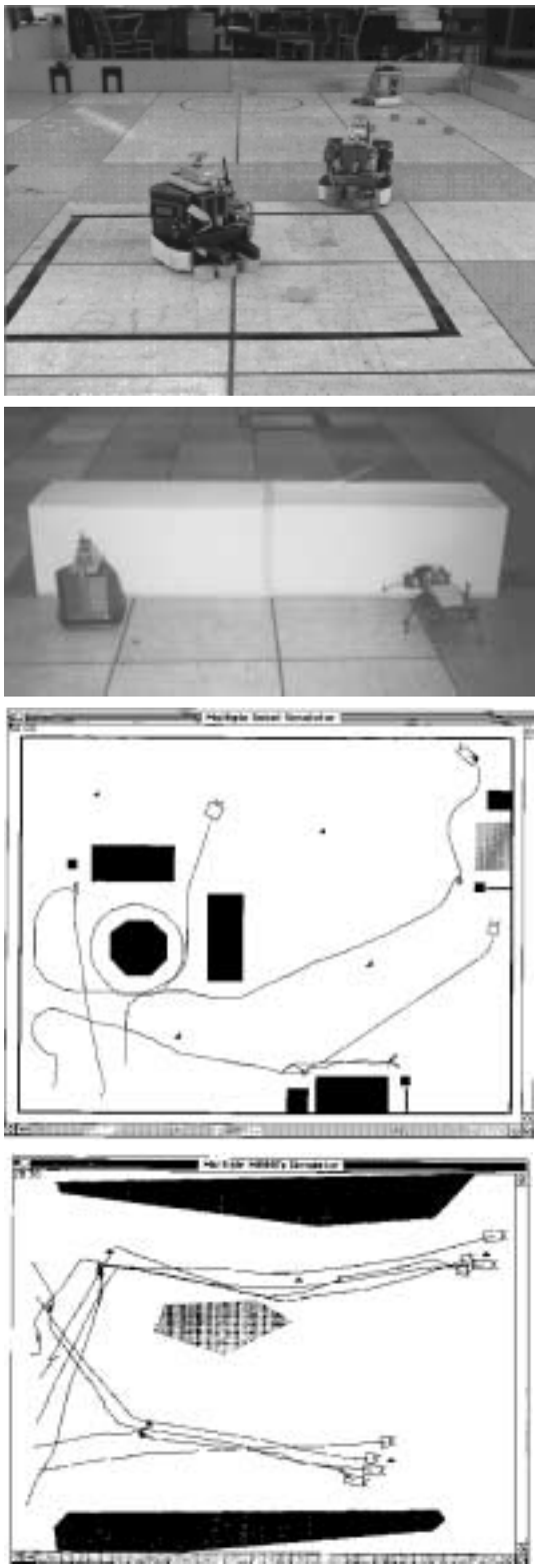


Figure 2: Implementations of the ALLIANCE architecture (on both simulated and physical robots). From top to bottom, these implementations are: "mock" hazardous waste cleanup, bounding overwatch, janitorial service, and box pushing.

Application domain	**# Robots**	**Metric description**	**Metric definition**
1. "Mock" hazardous waste cleanup	2-5 (P)	a. Time of task completion	t_{max}
		b. Total energy used	$\sum_{t=1}^{t_{max}} \sum_{i=1}^{m} e_i(t)$, where $e_i(t)$ is energy used by robot i through time t (m robots)
2. Box pushing	1-2 (P)	Perpendicular dist. pushed per unit time	$d_\perp(t)/t$, where $d_\perp(t)$ is $\perp$ distance moved through time t
3. Janitorial service	3-5 (S)	a. Time of task completion	t_{max}
		b. Total energy used	$\sum_{t=1}^{t_{max}} \sum_{i=1}^{m} e_i(t)$, where $e_i(t)$ is energy used by robot i through time t (m robots)
4. Bounding overwatch	4-20 (S)	Distance moved per unit time	$d(t)/t$, where $d(t)$ is distance moved through time t
5. Formation-keeping	4 (P & S)	Cumulative formation error	$\sum_{t=0}^{t_{max}} \sum_{i \neq leader} d_i(t)$, where d_i = distance robot i is misaligned at t
6. Simple multi-robot manipulation	2-4 (P)	Number of objects moved per unit time	$j(t)/t$, where $j(t)$ is number of objects at goal at time t
7. Cooperative tracking	2-4 (P) 2-20 (S)	Avg. number of targets observed (collectively)	$A = \sum_{t=1}^{t_{max}} \sum_{j=1}^{n} \frac{g(B(t),j)}{t_{max}}$, where $B(t) = [b_{ij}(t)]_{m \times n}$, ($m$ robots, n targets) $b_{ij}(t) = 1 \Longrightarrow$ robot i observing target j at t, $g(B(t), j) = \begin{cases} 1 & \text{if exists } i \text{ s.t. } b_{ij}(t) = 1 \\ 0 & \text{otherwise} \end{cases}$
8. Multi-vehicle production dozing	2-4 (S)	Quantity of earth moved per unit time	$q(t)/t$, where $q(t)$ is quantity of earth moved through t

Table 1: Summary of metrics used in ALLIANCE implementations. (In the second column, "P" refers to physical robot implementations; "S" refers to simulated robot implementations.)

Figure 3: Additional implementations of the ALLIANCE architecture. From top to bottom, these implementations are: cooperative manipulation, formation-keeping, cooperative tracking of multiple moving targets, and cooperative production dozing.

aimed at developing these higher-level metrics for the evaluation of robot team performance.

Acknowledgments

This article has been authored by a contractor of the U. S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes. Research sponsored in part by the Engineering Research Program of the Office of Basic Energy Sciences, U. S. Department of Energy. Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the U.S. Dept. of Energy under contract DE-AC05-00OR22725.

References

[1] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the Martha project. *IEEE Robotics and Automation Magazine*, 1997.

[2] T. Arai, H. Ogata, and T. Suzuki. Collision avoidance among multiple robots using virtual impedance. In *Proceedings of the Intelligent Robots and Systems (IROS)*, pages 479–485, 1989.

[3] T. Balch. Social entropy: An information theoretic measure of robot team diversity. *Autonomous Robots*, 8(3), 2000.

[4] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, December 1998.

[5] Tucker Balch and Ronald C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):1–25, 1994.

[6] J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario. Self-organizing collection and transport of objects in unpredictable environments. In *Japan-U.S.A. Symposium on Flexible Automation*, pages 1093–1098, Kyoto, Japan, 1990.

[7] Alexis Drogoul and Jacques Ferber. From Tom Thumb to the Dockers: Some experiments with foraging robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 451–459, Honolulu, Hawaii, 1992.

[8] Douglas Gage. Randomized search strategies with imperfect sensors. In *Proceedings of SPIE Mobile Robots VIII*, Boston, September 1993.

[9] David Jung and Alexander Zelinsky. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots*, 8(3), July 2000.

[10] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of AAAI-91*, pages 8–14, 1991.

[11] S. Marsella, J. Adibi, Y. Al-Onaizan, G. Kaminka, I. Muslea, and M. Tambe. On being a teammate: Experiences acquired in the design of RoboCup teams. In O. Etzioni, J. Muller, and J. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 221–227, 1999.

[12] Maja Mataric. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, 1994.

[13] David McFarland. Towards robot cooperation. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 440–444. MIT Press, 1994.

[14] L. E. Parker. Designing control laws for cooperative agent teams. In *Proceedings of the IEEE Robotics and Automation Conference*, pages 582–587, Atlanta, GA, 1993.

[15] L. E. Parker. *Heterogeneous Multi-Robot Cooperation*. PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, February 1994. MIT-AI-TR 1465 (1994).

[16] L. E. Parker. On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 1996.

[17] L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[18] L. E. Parker. Distributed control of multi-robot teams: Cooperative baton-passing task. In *Proceedings of the 4th International Conference on Information Systems Analysis and Synthesis (ISAS '98)*, volume 3, pages 89–94, 1998.

[19] L. E. Parker. Cooperative robotics for multi-target observation. *Intelligent Automation and Soft Computing, special issue on Robotics Research at Oak Ridge National Laboratory*, 5(1):5–19, 1999.

[20] L. E. Parker. Lifelong adaptation in heterogeneous teams: Response to continual variation in individual robot performance. *Autonomous Robots*, 8(3), July 2000.

[21] N. S. V. Rao. Terrain model acquisition by mobile robot teams and n-connectivity. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000)*, 2000.

[22] I. Rekleitis, G. Dudek, and E. Milios. Graph-based exploration using multiple robots. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000)*, 2000.

[23] D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 235–242, 1995.

[24] D. Rus and M. Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1726–1733, 2000.

[25] D. Stilwell and J. Bay. Toward the development of a material transport system using swarms of ant-like robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 766–771, Atlanta, GA, 1993.

[26] C. Unsal and P. K. Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1742–1747, 2000.

[27] A. Yamashita, M. Fukuchi, J. Ota, T. Arai, and H. Asama. Motion planning for cooperative transportation of a large object by multiple mobile robots in a 3d environment. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3144–3151, 2000.

[28] M. Yim, D. G. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 514–520, 2000.

Shared Autonomy and Teaming: A preliminary report*

Henry Hexmoor[αβ] and Harry Duchscherer[γ]

[α] Computer Science & Computer Engineering Department, Engineering Hall, Room 313, Fayetteville, AR 72701

[β]Center for Multisource Information Fusion University at Buffalo, Buffalo, NY 14260

[γ]University of North Dakota Grand Forks, North Dakota, 58202

ABSTRACT

We outline how an agent's shared autonomy considerations affect its interaction in a team. A unified model of acting and speaking will be presented that includes teaming and autonomy. This model is applied to the domain of satellite constellation. We introduce our simulator and outline our application of autonomy and teaming concepts.

KEYWORDS: Multi-agent Systems, Shared Autonomy, Agent Teams

1. INTRODUCTION

We have presented Situated Autonomy as a moment-by-moment attitude of an agent toward a goal and have argued that it is a useful notion in modeling social agents [6].

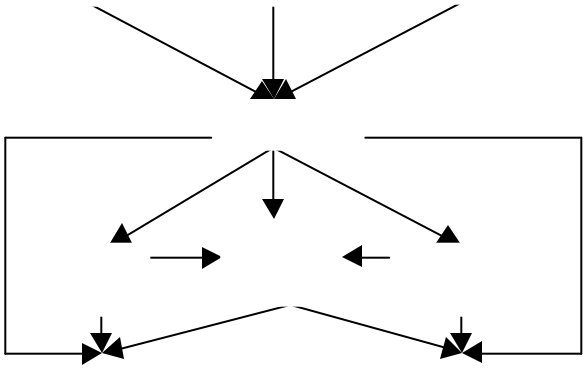


Figure 1 Action Selection

We argued that a combination of the nature and the strength of an agent's beliefs and motivations lead the agent to perceive one of the following: (a) the agent chooses itself to be the executor of the goal, (b) the agent delegates the goal entirely to others, (c) the agent shares its autonomy with other agents, or (d) the agent has a relatively small and undetermined responsibility toward the goal. Our focus in this paper is when the agent perceives shared autonomy.

Situated autonomy is an important part of an agent's action selection. Figure 1 shows a very simple action selection in Belief Desire Intention (BDI) paradigm and the role of situated autonomy. Along with goals and beliefs, we believe situated autonomy is used in the process of determining intentions. The process can be highly cognitive as in planning or less cognitive as in reaction generation. Enablers are the agent's perception of its own abilities, social factors, tools, and resources.

There are many accounts of starting or joining a team [2, 4, 10]. We favor the ingredients of intentional cooperation laid out by Tuomela: (a) collective goal or plan, (b) strong correlation among member's interest or preferences, and (c) having a cooperating and helping attitude.

We believe that in common situations, an agent's situated autonomy changes at a lot faster pace than its participation in a team. Once an agent perceives shared autonomy toward a goal, it may be inclined to recruit one or more agents to form a team. After a team is formed, the agent's degree of shared autonomy will change at the speed of perceived changes to the cognitive ingredients of situated autonomy. A recruited agent's degree of shared autonomy will be smaller than the recruiter's shared autonomy but after a team is formed will change to any level.

We are developing a model that unifies acting and speaking [7]. This model uses production rules to encode a conversational policy. A conversational policy is a modeling system that is designed to encode a set of conventions shared among a group of agents [5]. Such systems are generally called Normatives [1,10]. A prototypical agent follows the conventions of the group in communicating and sharing mental states. However, situated autonomies of each agent will individualize its interactions and allow it to deviate from the expected behavior.

* This work is supported by AFOSR grant F49620-00-1-0302.

We will present a model of conversational model in generic, non-BDI format. Each agent will personalize parts of conversational policy in its own BDI paradigm. A conversation policy is two types of simple production-like structures we will call transitions, shown below.

physical condition * spoken word/phrase * speak state
➔ speak state
physical condition * speak state
➔ spoken word/phrase

To model physical actions of an agent in reactive behaviors, we introduce two other types transitions, shown below.

physical condition * spoken word/phrase * act state
➔ act state
physical condition * act state
➔ act

A number of agents may share a unified model. For example, a group of agents may share a conversational policy. The shared model becomes their Norm. By entering a model and tracking the shared states, agents can synchronize their actions. Privately, each agent will consider transitions in terms of beliefs or goals, and intentions.

In the general model, physical conditions arbitrate among productions that provide alternative acts or words at a given state. However, each agent will have a personalized perception and interpretation of the physical conditions in terms of beliefs. We consider agents' situated autonomy and teaming consideration is determined by the agent's unique perceptions of the common physical conditions. Below we rewrite the transitions from an agent's perspective and add situated autonomy. 'Physical conditions' and 'spoken word/phrases it hears' are things about which an agent has beliefs. The states are the agent goals (or interchangeably desires). The 'physical act' or 'chosen word/phrase for communication' are the objects of an agent's intentions.

Belief(physical condition) *
Belief(spoken word/phrase) *
Goal(speak)
➔ Goal(speak)
Belief(physical condition) *
Goal(speak) *
situated autonomy
➔ Intention(spoken word/phrase)
Belief(physical condition) *
Belief(spoken word/phrase) *
Goal(act)
➔ goal(act)
Belief(physical condition) *
Goal(act) *
situated autonomy
➔ Intention(act)

The remainder of this paper is organized by working through an example of a unified model and how agents can personalize the physical conditions and consider teaming and changes in their Shared Autonomy. We will present a simulation of a constellation of satellites that can be tasked from ground. We will show our unified model and related issues of learning autonomy level using this application domain. We have not yet conducted experiments with situated autonomy and hence we consider this report a preliminary report.

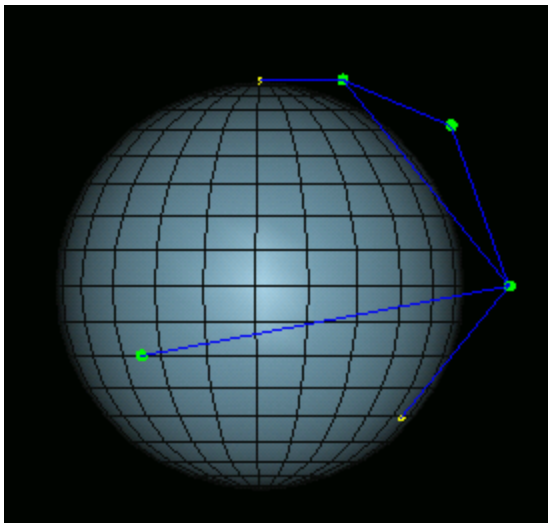


Figure 2. The Server's graphic screen

2. SIMULATION OF A CONSTELLATION OF SATELLITES

We have developed our own satellite simulator to illustrate our research ideas outlined in this paper. The simulator follows the principles of TechSat 21 [8]. SaVi is a similar software created at the Geometry Center at the University of Minnesota for the visualization and analysis of satellite constellations [3]. It has been used to simulate various satellite constellations such as Globalstar, Iridium, and Teledesic. SaVi differs from ours in that it is simply a simulator of orbital satellite constellations, and does not implement autonomy in its satellites.

Our simulator is composed of two primary modules; the server, and the agent. The server module handles the creation of all agent objects in the simulation and acts as a router to facilitate the passing of messages between various agents. There are two types of agents that can be created within this environment, satellite agents and ground station agents. These agents are implemented as objects and have similar capabilities, with the satellites having the additional ability to change their location within the environment. The server module is also responsible for the accurate

representation of all objects in the graphical environment (Figure 2).

The agent module contains the functional components of the agents. These components constitute the essence of the agent's purpose and functionality. Behavioral functions and autonomy states can be created and transitioned by accessing these module components through the use of behavior rules in the agent's behavior file. Behavior rules are comprised of conditional checks and assignment calls to the functional components in the form of simple production rules.

The satellite simulator was implemented using Mesa and supported by the collision detection routines which are part of the SOLID library package. The simulation is comprised of a central solid sphere surrounded by a wire-frame sphere to establish a latitude/longitude coordinate system. The sphere is currently scaled to represent the earth, and rotational velocity is approximately 120 times nominal. Graphically, the satellites are represented as green spheres with groundstations being yellow spheres on the planet's surface. The entire simulation can be rotated on any of the three axises. This allows for the simulation to be viewed from various prespectives. Additionally, any agent can be selected to be "tracked" in the simulation. This has the affect of centering the agent at the origin, with all other objects, including the planet, revolving around the agent. Blue line segments are used to show connections between satellites that have a line of sight communications capability, or connections between a satellite and a ground station (Figure 4). The SOLID library was used to make this determination, since the Mesa libraries do not directly support the detection of intersections between the connecting line segments and the planetary bodies. All satellites orbit at velocities which are appropriate for their altitude, with respect to a planet such as the earth.

The satellites and ground stations that orbit and reside on the planet are implemented as objects and have communication capability to other agents via message passing through the socket connection with the server. The position of each of these agents is determined by the data that is provided to the server in a text file. The text file contains only the most basic of information necessary to place the agent in the graphics environment of the server.

As each agent object is created, it reads a behavior file, which contains the rules that will govern its actions with respect to communication policy and physical actions that may be needed to achieve a desired goal. The format and examples of these rules is described in more detail in the next section.

3. TAKING 3 SCANS OF AN AREA

Assume the ground station will need three independent images of a given longitude and latitude from a given altitude. Let's call the task 3Image. The ground station issues the command to the nearest satellite and that satellite will be responsible to perform the task either by itself if no satellites are available. The satellite will complete the images itself taking one image in each orbit crossing the given location. If the satellite so decides it recruits other satellites to complete the task. Each of the recruited satellites may recruit another satellite. After recruiting one satellite, either satellite may decide to recruit a third teammate.

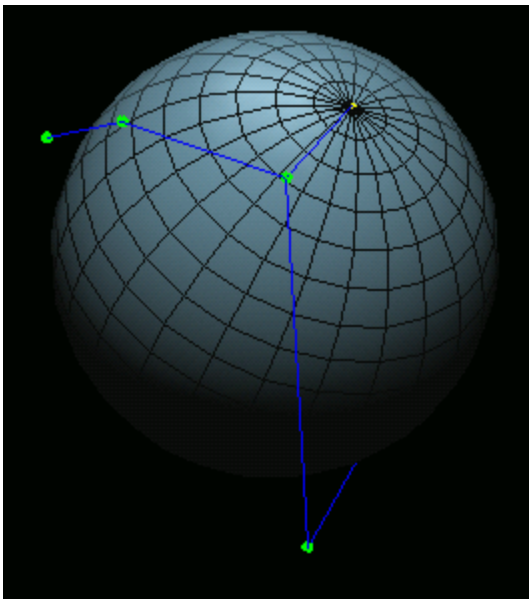


Figure 4. Communication lines

Here we will present a conversational policy that will govern interagent communication.

The following are the agent *speak states*:
0 – Start state
1 – Ground station has issued a command and a Satellite has received this message.
2 – A satellite has received and accepted the command.
3 – A second satellite has been contacted.
4 – The second satellite has accepted the command.
5– A third satellite has been contacted.
6– The third satellite has accepted the command and we now have a team.
7- Ground control has received the first image.
8- Ground control has received the second image.
9- Ground control has received the third image.
10- Success State

11- Failure State. This state occurs when any of the images are not received in a reasonable amount of time. State 0 is the start of 3-imaging.

The following are the set of available *words/phrases*:

S0 – Satellite agent says "Hello" to other agents to announce its presence, if it is currently idle.
S1 – Ground station issues a command 3Image [Longitude] [Latitude] [Altitude]
S2 – A satellite accepts command. The satellite says "Roger to 3Image"
S3 – Ground states acknowledges that a team leader has agreed to take the task and will now accept images by speaking "Ready to receive images".
S4 – A satellite recruits another satellites for 3Image. The satellite may say "Team 3Image?"
S5 – If a satellite accepts the request for being part of a team for 3Image, it may say "Willco".
S6 - If a satellite rejects the request for being part of a team for 3Image, it may say "Unable".
S7 – "bye" is spoken when a team member is no longer able to be part of the team.
S8 – "Downloading Image #1" is spoken when image #1 is downloaded to the ground Station.
S9 – "Downloading Image #2" is spoken when image #2 is downloaded to the ground Station.
S10 – "Downloading Image #3" is spoken when image #3 is downloaded to the ground Station.
S11 – "Received Image #1" is spoken when image #1 is received by the ground Station.
S12 – "Received Image #2" is spoken when image #2 is received by the ground Station.
S13- The ground station may say "Task Complete" when all three images are received.
S14- With an excessive silence, the policy ends unsuccessfully, "Task Aborted".

The following are the *physical conditions.* For each condition we note the agent that perceives it.

P0 – Start condition.
P1 - There is a need for 3Imgaing and a satellite is chosen for tasking. This condition is perceived by GROUND only.
P2 - Satellite is unable to participate in a team for one of two reasons: It is in danger or it has not yet finished its previous task. This condition is perceived by the SAT that is contacted to perform the task.
P3 – Satellite is able to take lead on a task and is available. This condition is perceived by SAT only.
P4 – Another satellite is detected that can potentially be a team-mate. This condition is perceived by SAT.
P5- Satellite is able to be a team-player. This condition is perceived privately by the SAT. All SAT agents privately perceive conditions P6-P10.
P6- An image has been collected.
P7- An image has been successfully collected and transmitted to the ground station.
P8- Two images are successfully collected and transmitted to the ground.
P9- Three images are successfully collected and transmitted to the ground.
P10– The chosen Satellite has received the command. Ground station is now ready to receive images. This condition is perceived by GROUND only.
P11- All the external conditions and instrumentation conditions for taking a picture are met.

In the following *speak state transitions,* each agent's type is noted by a "GND" for ground station or "SAT" for satellite. SATAVL, TIMEOUT, UNABLE, AND PICT are boolean conditions. SATAVL determines if a satellite agent is available (free of prior tasks and capable of taking on new a task) for the current agent. The availability is determined with respect to the satellite's current speak state and physical conditions. TIMEOUT holds if an excessive amount of time has elapsed since the last change in speak state. PICT indicates if the agent has any pictures that can be downloaded to the ground station. UNABLE denotes the satellite's propioception of being busy with a prior task or somehow being "out of service". PICT denotes the absence of such a condition. "SPK:*<destination>*" construct is used to specify to whom the spoken phrase is intended. CUR_AGNT is the agent most recently identified as available by the SATAVL check. The default CUR_AGNT is the speaking agent.

The following are the *speak-state transitions.*

P0*1*GND*S1*➔0

P3*0*SAT*S1➔1
P3*1*SAT*S3➔2
P4*2*SAT*S4➔3
P4*4*SAT*S4➔5
P4*3*SAT*S6➔2
P4*3*SAT*S4➔4
P4*5*SAT*S5➔6
P5*0*SAT*S3➔2
P2*3*SAT*S7➔0
P2*4*SAT*S7➔0
P2*5*SAT*S7➔0
P2*6*SAT*S7➔0
P2*7*SAT*S7➔0
P2*8*SAT*S7➔0
P4*4*SAT*S7➔2
P4*5*SAT*S7➔3
P4*6*SAT*S7➔4
P7*2*SAT*S11➔7
P7*4*SAT*S11➔7
P7*6*SAT*S11➔7
P6*2*SAT*S11➔7
P6*4*SAT*S11➔7
P6*6*SAT*S11➔7
P8*7*SAT*S12➔8

P6*7*SAT*S12→8
P7*7*SAT*S12→8
P9*8*SAT*S13→9
P6*8*SAT*S13→9
P7*8*SAT*S13→9
P8*8*SAT*S13→9
P10*0*GND*S2→1
P10*1*GND*S3→2
P10*2*GND*S8→7
P10*7*GND*S9→8
P10*8*GND*S10→9
P10*9*GND*S13→10
1*SAT*S14→11
2*SAT*S14→11
3*SAT*S14→11
4*SAT*S14→11
5*SAT*S14→11
6*SAT*S14→11
7*SAT*S14→11
8*SAT*S14→11
P10*1*GND*TIMEOUT→11
P10*7*GND*TIMEOUT→11
P10*8*GND*TIMEOUT→11

The following are the *speak transitions.* SA denotes the agent's level of situated autonomy.

P0*0*SAT*TIMEOUT→SPK:ALL*S0
P1*0*GND*TIMEOUT→SPK:CUR_AGNT*S1
P0*0*SAT*S4→P5
P3*1*SAT→SPK:ALL*S2
P10*1*GND→SPK:ALL*S3
P4*2*SAT*SA→SPK:ALL*S4
P5*0*SAT*S4→P2
P6*2*SAT*SA→SPK:ALL*S5
P7*2*SAT*SA→SPK:ALL*S8
P4*4*SAT*SA→SPK:ALL*S4
P2*4*SAT*SA→SPK:ALL*S7
P7*4*SAT*SA→SPK:ALL*S8
P2*6*SAT*SA→SPK:ALL*S7
P7*6*SAT*SA→SPK:ALL*S8
P2*0*SAT→SPK:ALL*S6
P2*3*SAT→SPK:ALL*S7
P2*5*SAT→SPK:ALL*S7
P2*7*SAT→SPK:ALL*S7
P2*8*SAT→SPK:ALL*S7
P8*7*SAT→SPK:ALL*S9
P9*8*SAT→SPK:ALL*S10
P10*7*GND→SPK:ALL*S11
P10*8*GND→SPK:ALL*S12
P10*9*GND→SPK:ALL*S13
P10*11*GND→SPK:ALL*S14

The following are the *act transitions.* "A" denotes an act, which in 3Imaging is taking a picture.

P11*2*SAT*SA→A
P11*4*SAT*SA→A
P11*6*SAT*SA→A

In addition to the conversational policy and action rules (above), we have designed rules for our agents to infer physical conditions based on exiting physical conditions and their current speak states and either (a) what they hear, (b) propioception of time or success of their own task (taking a picture), or (c) perception (availability of another satellite for teaming). We will consider these rules to be more domain oriented and intended for internal use of agents. Collectively, we will refer to these rules as domain rules.

The following are mainly based on hearing.

P1*0*GND*S2→P10
P2*0*SAT*S7→P0
P0*0*SAT*S1→P3
P4*4*SAT*S5→P3
P0*0*SAT*S3→P5

The following are mainly based on agent perception.

P0 * 0 * GND*SATAVL→ P1
P3*2*SAT*SATAVL→P4
P3*4*SAT*SATAVL→P4

The following are mainly based on agent propioception.

P1*1*GND*S1*TIMEOUT→P0
P5*2*SAT*PICT→P6
P4*6*SAT*PICT→P6
P4*6*SAT*PICT→P7
P6*2*SAT*PICT→P7
P6*4*SAT*PICT→P7
P6*7*SAT*PICT→P8
P7*7*SAT*PICT→P8
P6*8*SAT*PICT→P9
P7*8*SAT*PICT→P9
P8*8*SAT*PICT→P9
UNABLE →P2

4. USING CONVERSATIONAL POLICY

Agents can use the conversational policy for forming their beliefs, goals, and intentions. Each agent will apply the policy, action, and domain rules to new messages it receives. The following is our highest-level loop pseudo code for agent update.

```
For (agent; 1; numAgents)
    While (new receive message)
    {
        1. Determine SA
        2. For (rule; 1; numRules)
              If (rule applies)
                 a. Perform transitions
                    Use SA to resolve conflicts
                 b. Update beliefs and goals
        3. Perform the intention for speaking or acting
           within reaction constant
    }
```

Given a goal and the prevailing physical conditions agents constantly update their SA. SA is used in resolving conflicts in rules and in final decision of intention to be formed. Based on situated autonomy agents perform their picture taking or recruit other agents as teammates. The GND agent will note P0 or P1 (and form a belief) and will instantiate an instance of 3Imaging conversational policy. GND will maintain state 0 as its goal. Being in state 0 and having perceived P1, GND will use a speak transition to intend and then to issue S1. If the satellite (call it SAT1) has received the message S1 the speak state transition is used to reach state 1. GND and satellite SAT1 share the goal of being in state 1. SAT1 may perceive P3 and using a speak state transition to arrive at a desire to be in state 2 and also form an internal goal in achieving the command. GND does not determine P3 so it has no access to this perception. It however has access to the state transition that allows it to desire state 2. In state 2, SAT1 privately considers P3, P4, and P11 and arrives at a determination of situated autonomy. In 3Imaging, the lead agent once it reaches state 2, must consider exogenous physical conditions 3, 4, and 11 along with all agent endogenous factors to determine its autonomy. If it decides on shared autonomy, the agent must begin recruiting other agents as teammates. Otherwise, it will either do the task itself or delegate it to others.

If SAT1's decision favors a team formation, it uses a state transition to arrive at state 3 and forms a desire in it. Due to space limitation, we will not discuss the details of team formation. Since P4 is not shared with GND, it does not have the same belief. Let's call the second Satellite SAT2. SAT1 and SAT2 now share the desire to be in state 3. If SAT2 perceives P5, it will use a state transition and moves to state 4 and forms a desire in state 4 and the goal to be a teammate in 3Imaging. If SAT2 perceives P2, it will inform SAT1 and move back to state 2. SAT2 no longer has to want state 3. SAT1 will desire State 2.

For an agent that is recruited to be a teammate in state 4 it has already decided to have shard autonomy. It must consider its exogenous physical conditions 3 and 4 (P3 and P4) along with all agent endogenous factors to determine its autonomy in order to decide whether yet another teammate is needed. If it decides to recruit another agent it will move through states to state 6.

For an agent that is recruited to be a teammate in state 6 it has already agreed to have shard autonomy and since it is the third member of the team no other teammates are needed. Conditions P6-P9 may be perceive by either Satellite agent and all SAT agents share goals in state 7-11.

In the next section we will briefly discuss how autonomy will vary.

5. AUTONOMY MEASURES

Situated Autonomy depends on time, and strengths of belief and goal. [6]. Each agent reacts at different speeds. The times between sensing and acting is an agent's reaction constant and the optimal values can be learned. This greatly affects the agent's autonomy decision. Temporally, from the shortest reaction time to the longest, an agent's autonomy is based on it's pre-disposition, disposition, and motivation. Therefore, an agent's reaction constant is important. An agent's beliefs used in autonomy decision vary from weak to strong. An agent's goals are directed to self, other, or group. The goals vary in strength of motivation from weak to strong.

In 3Imaging, agents have different reaction constants and we are experimenting with the effect of slow versus fast reacting satellites. An agent's beliefs are about the physical conditions and the speak states and change in strength. The goals are about taking images and they vary based on the agent's prior commitment. If a Satellite agent has committed to a 3Imaging task, it might commit to yet another 3Imaging command if it senses that it can complete the task. After the first command, the motivation level for the goal is set to be less than for the first command. A combination of belief and goal degrees are used for determining SA.

As of this writing, our implemented system runs and images are gathered. However, we do not yet have situated autonomy experiments. We plan to compare runs of the system with different reaction constants. The autonomy levels in our agents will be learned as combinations of beliefs and goals. The metrics we will use for feedback are timeliness of images collected.

6. SUMMARY AND CONCLUSION

We have developed a production-style representational framework that unifies acting and speaking. Our representation extends conversational policy scheme. It explains how agents can use the shared normative models of conversational policy for forming private beliefs, goals, and intentions. We outlined a scheme for flexible teaming that uses the notion of situated autonomy.

We have implemented our model in the domain of constellation of satellites. Our system runs but we have not yet completed experiments with how timely team formation improves our system performance.

REFERENCES

[1] C.E. Alchourron and E. Bulygin, (1971). Normative systems, Springer Verlag, Wien.

[2] P. Cohen, H. Levesque, I. Smith, (1997), On Team Formation, In J. Hintika and R. Tuomela, **Contemporary Action Theory**, Synthese.

[3] G. Bergen (1998), **SOLID - Interference Detection Library,** Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. (http://www.win.tue.nl/cs/tt/gino/solid/solid2_toc.html)

[4] F. Dignum, B. Dunin-Keplicz, and R. Verbrugge, (2000), Agent Theory for Team Formation by Dialogue, In Proceedings of ATAL-2000, Boston.

[5] M. Greaves, H. Holmback, and J. Bradshaw, (1999). *What is Conversation Policy?* In Autonomous Agents (Agents-99) Workshop titled Specifying and Implementing Conversation Policies, Seattle, WA.

[6] H. Hexmoor, (2000a). A Cognitive Model of Situated Autonomy, In Proceedings of **PRICAI-2000** Workshop on Teams with Adjustable Autonomy, Australia.

[7] H. Hexmoor, (2000b). Conversational Policy: A case study in air traffic control, In Proceedings of International Conference in AI, **IC-AI-2000**, Los Vegas.

[8] TechSat 21: Advanced Research and Technology Enabling Distributed Satellite Systems, *Overview Briefing of TecdhSat 21*, http://www.vs.afrl.af.mil/vsd/techsat21.

[9] R. Tuomela, (2000), **Cooperation**, Kluwer Pub.

[10] A. Valente and J. Breuker, (1994). A Commonsense Formalization of Normative Systems, In Proceedings of the ECAI-94 Workshop on Artificial Normative Reasoning, J. Breuker (ed), Amsterdam, p. 56-67.

Real Time Distributed Expert System for Automated Monitoring of Key Monitors in Hubble Space Telescope

Reza Fakory, Ph.D.
Computer Sciences Corporation
mfakory@csc.com

Majid Jahangiri, Ph.D.
Computer Sciences Corporation
mjahangiri@v2pop.hst.nasa.gov

Abstract

A distributed expert system for monitoring the critical telemetry (the Key Monitors) of Hubble Space Telescope (HST) has been designed and developed. The Key Monitors Expert System (KMES) monitors the general health of the space craft operation through analysis of the Key Monitors data. KMES uses rule-based approaches and notifies operators/system engineers when it receives a limit violation from Front End Processor subsystem (FEP). The design of KMES is similar to the design of a previously reported system called "Expert System for Automated Monitoring" (ESAM) which was developed for HST [1]. However, KMES uses an approach different from ESAM's approach. ESAM was designed to monitor all telemetry mnemonics in a selected subsystem via establishment of tight limits for mnemonics. On the other hand, KMES has been designed to monitor the Key Monitors, providing notifications for out-of-limit conditions in accordance with documented operational procedures. Upon detection of an out of limit conditions, KMES analyzes data for contingencies. It fires appropriate rules to request associated engineering data from telemetry repositories. Subsequently, KMES sends e-mails and e-pages to notify the appropriate System Engineers (SEs) and Operators. The duration of a limit violation is monitored to eliminate transient faults. KMES logs all out of bound (limits) violations but only takes an action for each persistent violation. In addition, the distributed system design approach of KMES allows a pre screening of data variations to reduce the number of queued rules. Also, design of KMES was modified to include only selected part (sub-database) of a main database into KMES's subprocesses. The sub-database contains data associated with mnemonics that are used within the associated subprocess. This approach significantly reduced the required real-time execution time and the memory usage for the expert system.

KMES also allows the user to override any activated miscompare. This feature permits operators to adjust for known anomalies or changes in operational context. The system generates event messages to override actions; the events include a user login ID and the reason for the override.

Currently, KMES includes rules to monitor seven subsystems. KMES rules can be expanded to include rules for other subsystems. This paper describes the fundamental design and features of KMES. The results for a simulated scenario leading to failure of a Key Monitor and timely detection of the failure by KMES and ESAM are presented.

1 Background

The Vision 2000 Command and Control System (CCS) Product Development Team has been formed to reengineer the HST ground system [2,3]. The CCS ground system consists of several systems including System Monitoring & Analysis (SYM). Development of an expert system for telemetry monitoring, fault detection and recovery for the HST is one of the SYM's responsibilities.

Prior to design of KMES, the SYM group developed a real-time Expert System for Automated Monitoring (ESAM) [1]. The system was designed and developed to monitor the general health of the spacecraft and to detect faults within the Hubble Space Telescope (HST) via monitoring all telemetry mnemonics within a selected subsystem. It employs model-based/rule-based, hierarchical fault tree analysis with forward-chaining rule propagation to compare expected state values with true states. The system uses a custom-built neural network model and System Engineer (SE)-provided algorithms to dynamically derive the expected state values based on knowledge of real-time or stored spacecraft commands. During operations, real-time telemetry values (i.e., true states) are compared to the expected state values for

possible limit violations. The duration of a limit violation is monitored to eliminate transient faults. The system logs all miscomparisons but only issues a system event message for each persistent miscomparison. The persistence implementation approach significantly reduces the number of false miscompare messages. Currently, ESAM only includes rules and models associated with fault detection in Electrical Power System (EPS) of HST. Further expansion of ESAM for monitoring other subsystems of the spacecraft encountered two problems. First, for acquisition of telemetry data, from Information Sharing Protocol (ISP) into the expert system, a shared memory technique was employed to overcome synchronization between RTserver, the expert system server [1], and the ISP server. This design employed RTdaq, a COTS product from Talarian Inc. [6], that acquired data from shared memory and transferred data to RTserver. Further tests and analysis of results revealed that occasionally data was dropped during transmission from the shared memory to RTdaq. In addition, RTdaq did not have provisions for transmitting status flags that accompany the telemetry data from ISP, which indicate the general health and validation of the data. Second, modeling and development of rules, for incorporation of dynamic limits, required a significant amount of time from experts and system engineers with high level of expertise in the relevant subsystems of HST.
In order to overcome the first problem, it was decided to develop new modules with direct interface between RTserver and ISP via an existing middleware. For the second problem, it was decided to monitor the critical telemetry (the Key Monitors) and notify experts in accordance with Key Monitors documentation [7]. In this design, the limit values are constants that are defined in the Project Reference Database (PRD). The Front End Processor (FEP) subsystem of CCS detects limit violations for all monitors. KMES receives the Key Monitor values as well as the companion status flags from FEP. The status flag indicates limit violated Key Monitors. These new enhancements were incorporated into the design of KMES, and delivered as a part of a CCS Release delivery. The following sections describe the design features of the developed system.

2 Introduction

Expert systems are corner stones of knowledge Management [4,5] foundations and as such are designed to reduce dependency on humans and increase reliability of complex systems. For many cases, expert systems are simply a way to codify the explicit and sometimes tacit knowledge of experts (operators and system engineers) so it can be used to provide guidance and solutions for known problems. The real time Key Monitors Expert System (KMES) was designed and developed to automate the experts monitoring of the Key Monitors. In concept, KMES has been developed to automatically monitor the general health of spacecraft operation and notify operators and system engineers upon recognition of defined anomalies.

The Key Monitors are defined in the HST Contingency Plan document [7]. This document establishes a consistent and approved response to out of limit conditions or misconfigurations throughout mission operations. The out-of-bound limits have constant values defined in the Project Reference Database (PRD). In general, PRD includes two sets of limits namely yellow limit and red limit. For some Key Monitors mnemonics, the yellow and red limits coincide. In these cases, the response associated with red limit violation has priority over the response associated with the yellow limit violation. The FEP subsystem determines violated telemetry and sets a status flag, which accompanies the mnemonic value. For example, the FEP sets the companion status flag for a mnemonic to “L” when the telemetry value drops below the lower value of the red limit associated with the mnemonic. The following sections describe the developed system.

3 System Description

KMES is primarily a rule-based expert system. KMES subscribes and receives the Key Monitors mnemonic values and the companion status flags from ISP. Most of KMES rules are simple and the hierarchy is shallow. However, the required actions for some limit violations are contingent on configuration or statuses of other equipment. Therefore, the rules associated with these limit violations have hierarchical levels. Upon detection of a violation, KMES looks for persistence of the violation. If the mnemonic’s value remains beyond limit boundaries, for a time greater than the persistence period, KMES

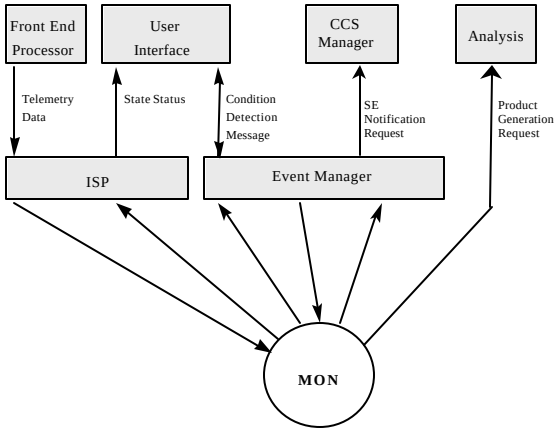


Figure-1 KMES External Interfaces

fires the associated contingency rules. As a part of actions within these rules, KMES sends requests to the Analysis subsystem for historical data products. Figure-1 shows KMES external Interfaces. KMES specifies the start time as well as the stop time for the requested data. Format of the requested data and type of the requested historical data are stored in ASCII files that are accessible to the Analysis subsystem. The generated historical data products are stored in a directory accessible to operators and system engineers. The system engineers (SEs) may use the data products to further analyze potential problems use the products.

4 KMES Architecture

Figure-2 shows process architecture for KMES. KMES consists of sub-processes for data communication as well as subprocesses for evaluation and reporting of the state of the spacecraft. The main processes are:

- RTD (Receive Telemetry Data)
- MGS (Manage States)
- REF (Respond to Events and Faults)
- PMD (Publish Monitoring Data)
- RMD (Route Monitoring Data), the RTserver

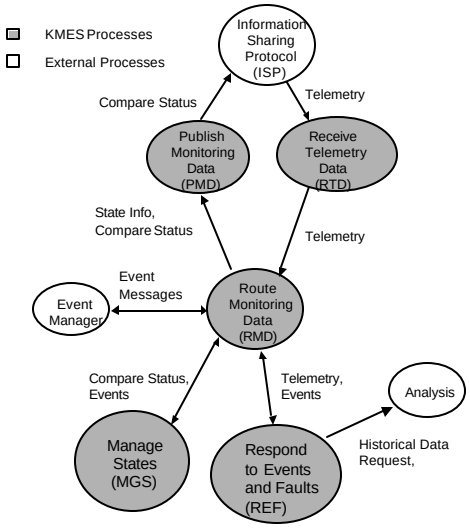


Figure-2 KMES Distributed Architecture

Originally, KMES employed a single database where all of KMES's processes included a copy of the data-base. However, this approach caused excess increase in the size of run time memory usage when KMES was expanded to include all seven subsystems of HST. Therefore, the design of KMES was modified to reduce the size of memory usage. In this new approach, every subprocess of KMES includes part of database that contains data related to mnemonics which are referenced or used within the subprocess. The following sections briefly describe function and features of each sub-process within KMES

4.1 Receive Telemetry Data (RTD)

The RTD process receives change-only data from the ISP server. RTD sends this data to the other KMES processes via the RMD process. Originally this subprocess employed RTdaq (a commercial product) and shared memory approaches for synchronization between RMD and ISP. However, it was found that occasionally data was dropped out during transmission between shared memory and RMD. In addition, RTdaq did not have capabilities to transmit status flags, which accompany the telemetry data. ISP sends the status flags as a part of data throughout the CCS subprocesses. These status flags indicate the status of data and they are set by the FEP subsystem within the CCS. A blank status flag indicates that the data is valid. At this time, ISP sends telemetry data with status flags that are set to nine possible values, one at a time. The status flag values are prioritized, four of these values indicate that the telemetry value is beyond pre-specified limits as defined in the PRD. The remainders of the status-flag values indicate if there has been a problem with data conversion or data transmission. The RTD subprocess was enhanced to eliminate the use of RTdaq as well as the shared memory approach. The enhanced version constructs custom designed data-packets that are in RTsmartSockets format. The packets contain changed only data and are sent to RTserver (RMD) for distribution to subprocesses within KMES.

4.2 Manage States (MGS)

The MGS process receives real-time telemetry data from ISP and generates compare status associated with each received Key Monitors mnemonic. The compare status indicates if there is a miscomparison (corresponding to a limit violation). This subprocess sets compare status in accordance with values of status flags that accompany telemetry data received from ISP. A compare status mnemonic may take four different values for a miscomparison corresponding to four possible ways of limit violations:

a) Yellow Low;
b) Yellow High;
c) Red Low;
d) Red High.

Yellow limits are warnings as specified by system engineers. Red limits are typically for serious violations associated with hardware limitations. MGS sends all compare status changes along with their status flags and time stamp to REF subprocess via RMD.

4.3 Respond to Events and Faults (REF)

REF includes all rules associated with limit violated Key Monitor mnemonics. Upon receipt of a miscomparison associated with Key Monitors from MGS, REF tracks the miscomparison for a pre-specified period of time (persistence time). If the limit violation persists, then REF fires the appropriate rules and sends appropriate historical data request with specified start time and stop time to the Analysis subsystem. REF also sends an event that indicates detection of the anomaly. The event message also indicates how soon the requested data product will be available for access by SEs or operators. The Analysis subsystem receives information for the data request from REF and retrieves the historical data in accordance with the format specified by SE(s) and Operators. The list and format of the data request are stored in specially designed files called "Historical Request Definition Files".

4.4 Publish Monitored Data (PMD)

The PMD process receives state data consisting of mnemonic's name, value, and time stamp from MGS (via RMD) while publishing this data to ISP. Along with this data there is a status flag indicating the override status of the mnemonic's value. The status flag indicates whether the user

has overiden the mnemonic value within KMES or that the mnemonic value is derived by KMES.

4.5 Route Monitoring Data (RMD)

The RMD process consists of a real time RTserver. The process receives and routes data and event messages within KMES's processes.

5 KMES Characteristic Features

KMES consists of a group of distributed processes that communicate through a middleware layer. This modular design has many advantages such as maintainability and flexibility in where and how these processes are executed. If one process is overloading system resources, it can be relocated to another host machine. Among other advantages, KMES employs a distributed system approach to facilitate:

a) Change Data only executions
b) Maintenance simplification

This approach provides a capability to queue only those rules that are affected by the status of a mnemonic. In this way, only the rules that have to send a historical data request and e-mail or e-page will be fired and the rest of the rules will not be examined until later times when a status change affects them.

6 Results

KMES has been developed with rules associated with actions that are required when a Key Monitor has violated its limits. Currently, Rules associated with the following seven subsystems of HST have been implemented:

- Data Management Subsystem (DMS)
- Electrical Power Subsystem (EPS)
- Instrumentation & Communication Subsystem (I&C)
- Optical Telescope Assembly (OTA)
- Pointing Control Subsystem (PCS)
- Safing Subsystem (Safing)

The following section discusses the results obtained from operation of KMES during a simulated anomaly. For comparison, the results of the previously designed system, ESAM, for the same simulated anomaly is also demonstrated. As it was mentioned earlier, ESAM detects anomalies by comparing the engineering telemetry received from HST with some internally generated expected values. ESAM analyzes the discrepancies between the true and expected states to determine if an anomaly actually exists. Therefore, ESAM uses some tight and dynamically calculated limit boundaries. In contrast, KMES depends on some predefined and fixed limit boundaries.

The following section, compare the results from ESAM and KMES for a simulated scenario leading to failure of a sensor in the Electrical Power System of the spacecraft.

7 Scenario

The test scenario was designed to examine the rule executions resulted from an anomaly associated with one of the HST batteries. The spacecraft is equipped with six batteries. If only four batteries are nominal then the entire battery system is considered acceptable for normal operation. Previously captured data from a routine spacecraft orbit was fed into the HST simulator. The simulator was started in play back mode with continuous data feed. Figure-3 shows voltages for the first and the second batteries of the spacecraft, respectively. Figure-4 shows the currents associated with the first and second batteries. Figure-3 and Figure-4 also depict the high-expected limit and the low-expected limit calculated by ESAM. For comparison, Figure-3 also shows the constant limits used for Key Monitors out of bound violations. As shown, the constant limits are normally wider than the limits calculated by ESAM. The results for battery one demonstrates that the system was in normal operation until time 23:05, at this time (point A) a ramp down sensor anomaly was simulated into the telemetry data for voltage of the first battery. Figure-3 shows that within about 4 minutes and 40 seconds (point B) into the incident, the battery voltage fell below the low limit as calculated by ESAM. However, Figure-3 shows that after 9 minutes and 30 seconds into the anomaly, the battery voltage fell below the constant limit used by the FEP. At this time KMES received an associated status flag from FEP that indicated the limit violation. Therefore, KMES queued the rules associated with the battery anomaly and sent appropriate

notifications and historical data requests when the limit violation persistence was satisfied. The results show that ESAM detected anomaly within about five minutes after initiation of the anomaly. However, KMES sent anomaly notifications within ten minutes after initiation of the anomaly.

8 Future Work

A well-structured distributed expert system to monitor the Key Monitors of the Hubble Space Telescope has been developed and delivered. The results for the first release of this system are presented. The following highlights some of the items sought for improvement and further enhancements of the monitoring system:

a) a variable persistence time for each or subset of the Key Monitor mnemonics;
b) retrieve appropriate operating procedure(s) for response associated with an anomaly;
c) track violation changes from yellow limit boundary into red limit violation boundary;
d) accept a user-defined periods for

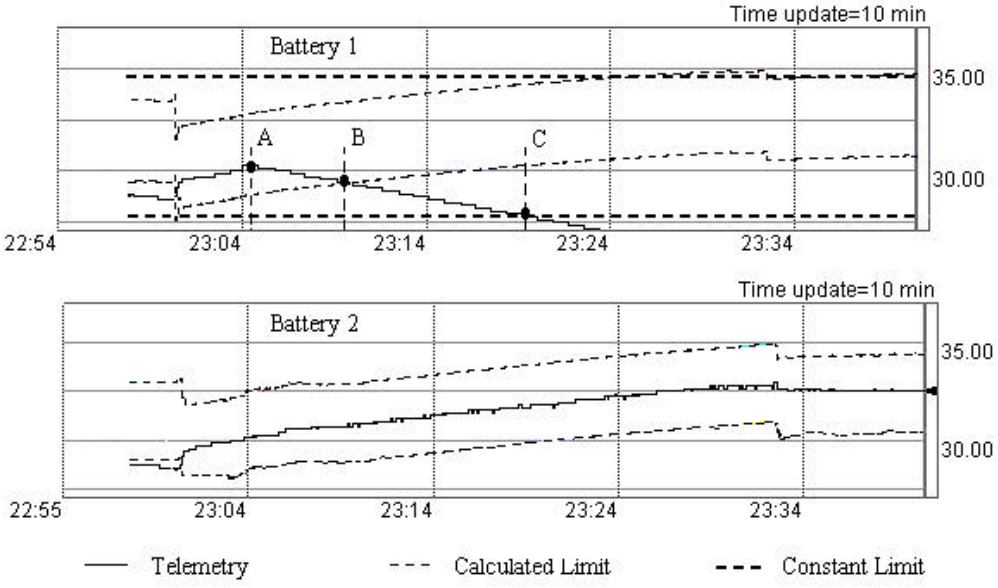


Figure-3 Results For Battery Voltage

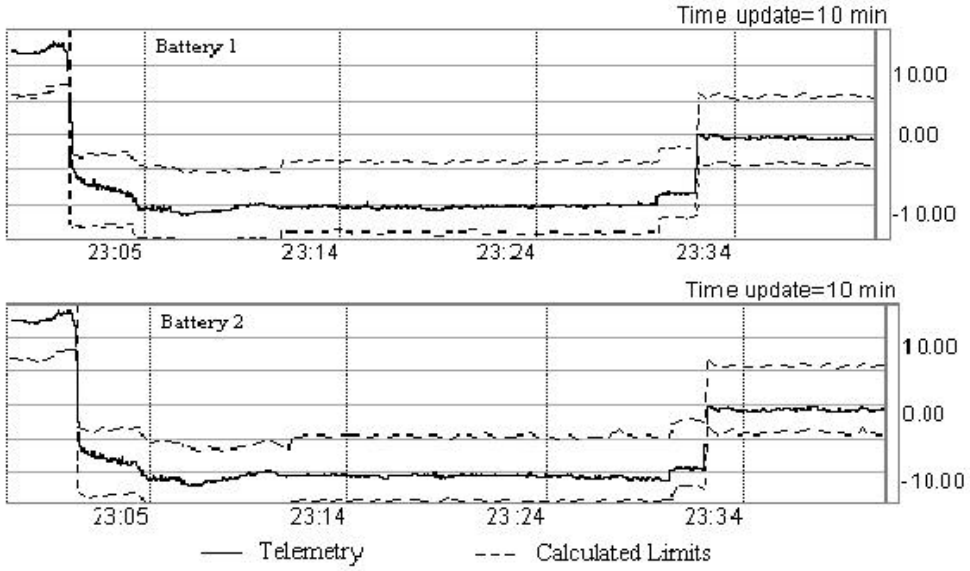


Figure-4 Results For Battery Current

notifications frequency associated with each of the limit violations;

e) GUI editor interface for visualization and graphical editing of rules.

9 Conclusion

A distributed expert system, KMES, for notification of anomaly and initial response (i.e., request associated engineering data for analysis) has been developed. The results of KMES for a simulated failure has been compared with similar results obtained from a previously designed expert system, ESAM. KMES uses the results of anomaly detection with constant limit values while ESAM calculates the expected limit boundaries. The results for detection of a sample sensor failure by the two systems are demonstrated. It was found that when calculated limits are employed then anomaly might be detected earlier than when constant limits are used for detection of the anomaly. However, notification of limit violations based on constant and established limits provides facilities for timely development of KB rules and execution of approved notifications.

10 Nomenclature

CCS	Command and Control System
ESAM	Expert System for Automated Monitoring
FEP	Front End Processor
ISP	Information Sharing Protocol
KMES	Key Monitors Expert System

11 References

[1] "A Real-Time Expert System for Automated Monitoring of the Hubble Space Telescope," R. Fakory and E. Ruberton, Intelligent Systems Conference, Gaithersburg, MD, September 1998.

[2] Consolidated HST Associated Mission Products (CHAMP) Contract, NAS50000.

[3] "Re-engineering of the Hubble Space Telescope (HST) Reduce Operational Costs (PartII)," M. Garvis, K. Lethonen and W. Burdick, internal report.

[4] "Knowledge Management Handbook," Jay Liebowitz, CRC Press 1999.

[5] "Development and Deployment of a Rule-Based Expert System for Autonomous Satellite Monitoring," L. Wong, F.Kronberg, A. Hopkins, F. Machi, P. Eastham, Astronomical Data Analysis, Vol 101, 1996.

[6] Talarian Corporation, support@talarian.com.

[7] "Hubble Space Telescope (HST) Contingency Plan document," Vol2, Lockheed Martin Missiles & Space (LMMS), report No. LMSC/P061924, 1997.

EVALUATING PERFORMANCE FOR DISTRIBUTED INTELLIGENT CONTROL SYSTEMS

Wayne J. Davis
General Engineering, University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA

ABSTRACT

This paper provides a new framework for the distributed intelligent control of complex systems. The behavior of a given subsystem as it interacts with other subsystems is explored. The inherent limitations associated with distributed planning and control procedures are revealed. These limitations further limits one ability to evaluate system performance. Knowing these limitations, allows one to seek improved procedures for managing complex systems, which should also lead to improved system performance.

1. INTRODUCTION

Measuring system performance inherently represents a subjective task, beginning with the definition of the considered system. The decision of what system elements will be included within the considered system is arbitrary. Moreover, excluding elements from the considered system does not eliminate the potential for these elements to interact with the considered elements. Rather, such interactions become inputs to the considered system, whose values cannot be controlled. The definition of the considered system necessarily constrains the performance of the system because one must relinquish control of these environmental inputs.

Any performance criteria employed to evaluate a system must be based upon system variables that can be measured and controlled. Hence, the scope of the considered system inherently constrains the type of performance evaluations that can occur. Often there are multiple criteria to be considered, which necessitates compromise among the appropriate criteria. Compromise requires a subjective prioritization among the considered criteria, making absolute performance evaluations nearly impossible to achieve.

It becomes difficult to analyze and manage a complex system as a single monolithic entity. Complex systems are better represented as systems-of-systems. Again, the definition of the included subsystems is arbitrary. Furthermore, each included subsystem will have its own state and control variables. These variables again constrain which performance criteria can be considered by each subsystem. What often emerges is a collection of subsystems whose behaviors are characterized via different performance criteria.

Even if a monolithic specification for the system's planning and control problems can be made, there are still shortcomings, expecially since the monolithic approach ignores the system-of-systems nature. Monolithic specification do not capture the multi-resolutional nature of complex systems where given subsystems address system variables at different levels of detail and on different time scales. Monolithic approaches do not scale well. For large-scale systems, monolithic approaches become impossible to implement.

On the other hand, distributed planning and control introduces other problems. Today's distributed planning and control technologies do not capture the true nature of the distributed planning and control requirements for complex systems. Most decomposition procedures for distributing planning still assume that a monolithic planning problem exists (see Lasdon [1] and Wismer [2]). In general, this monolithic planning problem cannot be defined; and even if it could, its complexity would be well beyond the scope of problems that can be addressed with available decomposition procedures. Decomposition algorithms further seek an optimal solution to the monolithic planning problem. However, the relationship of optimal planning at the subsystem level toward the optimal planning for overall system within which it resides is simply not understood. Today, we do not know how to coordinate the planning at a subsystem in order to insure global optimality for the overall system within which the subsystem resides.

Control is essential to implement plans. Again, the current distributed control technologies are limited. Perhaps the most common distributed control procedure is the slow-fast decomposition (see Kokotovic et al. [3]). Slow-fast decompositions certainly can address situations where the desired response is known. They usually assume that an aggregated description for the overall response is known over an extended horizon, which includes the

current time. Subsystems then manage the detailed description of this same trajectory over a shorter horizon which again includes the current time. This process continues where each subsystem addresses more detail over an even shorter horizon beginning with the current time. Implicitly, a monolithic control policy has been developed in that one assumes that the desired aggregate response is known over the entire time horizon.

Distributed intelligent control (distributed planning and control) approaches do not permit a monolithic description of the desired system trajectory. Rather, the system trajectory evolves as a collective response of several subsystems considering different temporal horizons and system elements. The planned response and the associated implementing actions evolve with time. Neither the monolithic planning or control problems are ever stated or solved. It is obviously difficult to manage such systems. Even more difficult is projecting their performance.

I revisited the distributed planning and control problem last year. My desire was to define what a distributed planning control system could accomplish. All the basic principles, including optimality and controllability, were set aside. The goal was to determine how a subsystem could address its assigned planning and control responsibilities while effectively interacting with other subsystems. Subsequently, the coordination of the interactions among the entire ensemble of distributed planning and control systems in order to provide an effective overall system response had to be addressed Testing effectiveness became a concern given the inherent inability to define the overall system problem as it continued to evolve in time.

This paper provides a brief discussion of the basic discoveries arising from this rapprochement. The fundamental principles of optimality and controllability have been reexamined and mathematical proofs/arguments do exist for the inherent limitations. Unfortunately, space limitations prevents me from providing these mathematical arguments. Instead, this paper will provide basic discoveries only. The paper first investigates how subsystems interact with each other. Next, the comprehensive nature of the overall system response arising from these interactions is addressed. Finally, the inherent limitations upon planning and control will be itemized. These limitations fundamentally impact one's ability to manage and project system performance. They must be addressed.

Figure 1: Basic interactions for a subsystem.

2. Fundamental Concepts

We begin our development with two basic assumptions:

- Most complex systems can be represented as a collection of subsystems that interact with each other. That is, complex systems are actually systems of subsystems.
- Each subsystem has a purpose, which it fulfills by executing tasks. Furthermore, the tasks that each subsystem can execute are related to the tasks that other subsystems can execute.

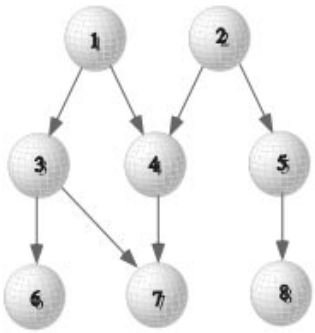
Consider a single subsystem. Its associated control inputs include (see Figure 1):

- Endogenous control inputs that it generates in order to implement its planned response.
- Assigned goals from other subsystems.
- Feedback information from other subsystems.
- Exogenous inputs from a subsystem's environment.

For a given subsystem, the assigned goals and feedback information might also be considered as exogenous inputs because these are generated by other subsystems. However, a given subsystem can determine which goals it will accept. The goals that it assigns to other subsystems will also influence their behavior and subsequent feedback information. Thus, only environmental inputs cannot be influenced in any manner by a given subsystem.

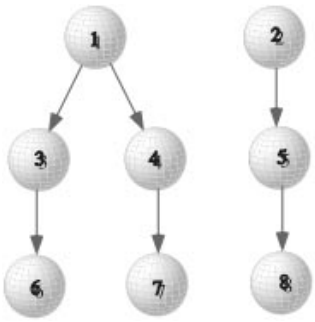
Should the subsystem have an option to accept or reject a goal? We believe that such an option is essential in order to insure that the recipient subsystem can feasibly respond to the goal. If one subsystem cannot satisfy its assigned goals, then the subsystem cannot respond in a feasible manner and the ability to control the subsystem is diminished or eliminated.

The assignment of goals to another subsystem represents one type of output that can occur as a subsystem responds to its control inputs. In addition, the given subsystem must provide feedback information to any other subsystem from which it has accepted a goal. Finally, the



Node	Assignors	Type
1	---	Creator
2	---	Creator
3	1	Coordinator
4	1, 2	Coordinator
5	2	Coordinator
6	3	Process
7	3, 4	Process
8	5	Process

Figure 2: Network representation of subsystem relationships.



Node	Assignors	Type
1	---	Creator
2	---	Creator
3	1	Coordinator
4	1	Coordinator
5	2	Coordinator
6	3	Process
7	4	Process
8	5	Process

Figure 3. Hierarchical system(s) where each subsystem (node) has at most one Assignor.

subsystem may also generate outputs that act upon the system's environment.

Typically, when one seeks to coordinate subsystems, one employs hierarchical based notions of supervisors (supremals) and subordinates (infimals). Hierarchies, right or wrongly, have been the subject of much recent criticism. In this paper, our desire is to provide a neutral atmosphere for discussing such coordination concerns.

We define the Assignors as the set of controllers that can assign goals to a given subsystem. Acceptors are the set of subsystems to which a subsystem can assign goals. Figure 1 depicts the proposed relationships among the subsystems.

There are two special situations. If the set of Assignors for a given subsystem is empty, then the subsystem receives only exogenous inputs from the its environment and feedback information from its Acceptors. We refer to such a subsystem as a Creator because it generates goals only, and does not accept any goals from any other subsystem. Every system model requires at least one creator. However, Creators are generally artificial constructs resulting from the modeling process. That is, the Assignors for a Creator are assumed to be outside the scope of the modeled system. Thus, goals coming from these external subsystems, or implicit Assignors, are viewed as inputs to a Creator from the system's environment.

If the Acceptors set for a given subsystem is empty, the subsystem is a Process. Processes can accept and process goals, but they cannot reassign their goals to any other subsystem. Hence, Processors can only accept inputs from their Assignors and the system's environment. In response to these inputs, they generate outputs upon the environment and provide feedback information to their Assignors.

We can graphically represent the proposed system structure (see Figure 2). We first define a node for each subsystem. We then employ directed arcs from a given subsystem's node to each node within its Acceptors set. Finally, from each subsystem node within a given subsystem's Assignors set, we draw a directed arc to the node for the given subsystem. Using network terminology, the Creator(s) become the source(s) to the system network while the Processes are the sinks.

In general, there can be more than one path from a given Creator to a given Process. (In Figure 2, there are multiple paths from node 1 to node 7.) However, there need not be a path from every Creator to every process. (In Figure 2, there is no path from node 1 to node 8.) In the special case where the number of elements in each subsystem's Assignors set is less than or equal to one, the representative system network becomes a tree and represents a conventional hierarchy (see Figure 3). If there is more than one Creator in the hierarchical case, then the overall system must be represented as a set of disjoint hierarchies that do not interact with each other (see Figure 3).

Although the potential for loops can exist within a system's network, loops should not exist from the conceptual point of view. Later we will show that the detail considered by an Acceptor is greater than its Assignor. We will also show that the planning horizon for any Acceptor should be less than that of the Assignor. Loops could occur when an Assignor for a given subsystem is also contained in the given subsystem's Acceptors set. However, if a subsystem is simultaneously contained within the Acceptors and Assignors sets for another given subsystem, then the simultaneous Acceptor/Assignor must be less detailed from the system to which it assigns goals and more detailed than the same system from which it accepts goals. Obviously, the two implied relationships are contradictory.

Therefore, we may conclude that the network representation of the relationships among subsystems for all meaningful systems must be a directed acyclic network (containing no loops).

The reader should note that we have spoken of goal assignments rather than tasks, as mentioned earlier. We assume that a goal can contain a task. Moreover, a goal can also describe how an assigned task should be executed. For example, the Assignor might request that the task be completed by a given time or completed at minimum cost.

3. Toward an Integrated Approach

No (sub)system can generate an optimal response when acting as an independent agent. A given subsystem's response is dependent upon its goals and the subsequent response of the subsystems to which it has assigned goals. Furthermore, one cannot demonstrate that the collective response arising from the coordinated interaction among all its subsystems is optimal because we have not (and cannot) define the overall problem.

Because no subsystem can respond independently from the other subsystems, it follows that each subsystem must constantly interact with other subsystems: its Assignors and Acceptors. However, a given subsystem's interactions with an Assignor are fundamentally different from its interactions with an Acceptor. Each subsystem considers a time interval and a level of detail that differs from those of its Assignors and Acceptors. Each subsystem must move from its current state to a specified goal state while responding to any external inputs from the overall system's environment and any peculiarities that arise in its own dynamic evolution.

Let us consider the interaction of a given subsystem with its Assignors. A given subsystem can only address its behavior over an interval. Nevertheless, the way in which a subsystem responds within a time interval can affect the future behavior of the entire system beyond the considered time interval. The problem is that the given subsystem is incapable of assessing these future consequences beyond the time interval that it considers. The subsystem must instead rely upon the subsystems contained within its Assignors set to make such assessments. In performing this function, each Assignor considers the future in order to specify goals for the given subsystem. The subsystem receiving the goals employs those goals in order to define its desired final state at then end of its planning horizon.

Similarly, most subsystems are also limited by the level of detail that they can consider. In order to affect the more detailed responses that are required to meet its assigned goals, each subsystem assigns goals to its Acceptors. Thus, as each Acceptor addresses an assigned goal, it provides a more detailed system response on behalf of the subsystem that assigned the goal. The subsequent feedback information provided by the Acceptor during its execution of an assigned task assists the Assignor in assessing the beginning state for its planning horizon. (Later we will demonstrate that this beginning state for a subsystem's planning horizon *cannot* be the current system state. It must always be a projected future state from which the given subsystem will attempt to a desired final state.) Two extremes, or boundary conditions, for a given subsystem's planning/control problem have now been specified. Its included planning and control (intelligent control) capabilities then guide the given subsystem from its projected initial state toward the desired final state while responding to forecasted environmental inputs and other peculiarities of the system response.

Every element of the subsystem's planning/control problems changes with time. The subsystem's estimate of its initial state changes as its Acceptors execute their assigned tasks. The subsystem's goal changes with time as its Assignors respond to feedback information that the subsystem provides. Finally, the forecasts for the subsystem's future interactions with its environment must be constantly updated.

We can now define three basic functional requirements for each subsystem's intelligent controller. These include:

Task Accepting: The intelligent controller must interact with the intelligent controllers that manage each subsystem within its Assignors set. The purpose of this interaction is to define new goals and to update current goals. Each assigned goal specifies at least one task to be addressed along with a set of constraints. Because an Assignor addresses the system in a more aggregated sense than the subsystem that accepts the task, the Task Accepting function must decompose the assigned tasks into subtasks. In addition, the execution constraints accompanying each accepted task must also be reformulated in order to specify appropriate (or consistent) constraints for each defined subtask.

The task decomposition and the associated constraint specification comprise a goal decomposition process. This goal decomposition must guarantee that the

accepted goals can be satisfied given the accepting subsystem's current state. The Task Accepting function is also responsible for continuously updating the projected response of the subsystem as feedback information to each Assignor. Remember, however, that an Assignor considers the system response in an aggregated manner. Thus, the Task Accepting function must summarize its projected response in order to provide the estimated performance statistics that can be understood by its Assignor.

Task Assigning: After the assigned goals are decomposed, the resulting subtasks and their associated execution constraints must be reassigned to the subsystem's Acceptors. In making the subsequent goal assignments, the Task Assigning function will employ the selected control law that implements the subsystem's current plan. The Task Assigning function also monitors feedback information from each Acceptor to which it has assigned a goal. Using this feedback information, it projects the future performance of the subsystem as it continues to execute its assigned goals under the selected control law. This projected response is then employed by the Task Accepting function within the same intelligent controller in order to provide feedback information to the subsystem's Assignors.

Performance Improvement: The system now has an estimated current state as well as a projected response as it implements its current goals under the planned response and enabling control law. The Performance Improvement Function continuously seeks a better control law for implementing the subsystem's assigned goals. Remember, however, that every element of the control problem is dynamic and uncertainties do exist. Given this reality, closed-loop control laws inherently perform best because they can tailor their response to the system's current state. It is also desirable to employ predictive control procedures whenever the current control action depends upon both the system's current and predicted future state. Whenever a new control law is selected, it is forwarded to Task Assigning function for implementation.

4. The Fundamental Principles of a Coordinated Response

This section addresses the basic system response. Figure 4 provides a primitive schematic for the multi-resolutional behavior of these systems. Let t_0 represent the current time that advances with real-time. We then divide the future time axis into several intervals, including $[t_1, t_2)$, $[t_2, t_3)$, $[t_3, t_4)$ and so forth. Note that we have not yet included a time interval between $[t_0, t_1)$ or $[t_4, ¥)$ for reasons to be discussed later. In Figure 4, the entire state vector has been projected as a single value upon the y-axis. This state trajectory is further divided into segments: one segment for each time interval specified above. Let us assume that each segment corresponds to the trajectory for a given subsystem operating under the control of its intelligent controller. Considering the subsystem associated with the state trajectory on the interval $[t_3, t_4)$, its Assignors manage the state trajectory beyond t_4, while its Acceptors manage the state trajectory on the interval $[t_2, t_3)$.

Suppose we view each component of the state trajectory as a sophisticated "Slinky." The multi-resolutional nature of the systems implies that size and length of each "Slinky's" spring gets smaller and shorter as its associated time interval approaches t_0. Now let us further assume that two adjacent "Slinkies" are attached to each other and that the boundary conditions must match at each junction.

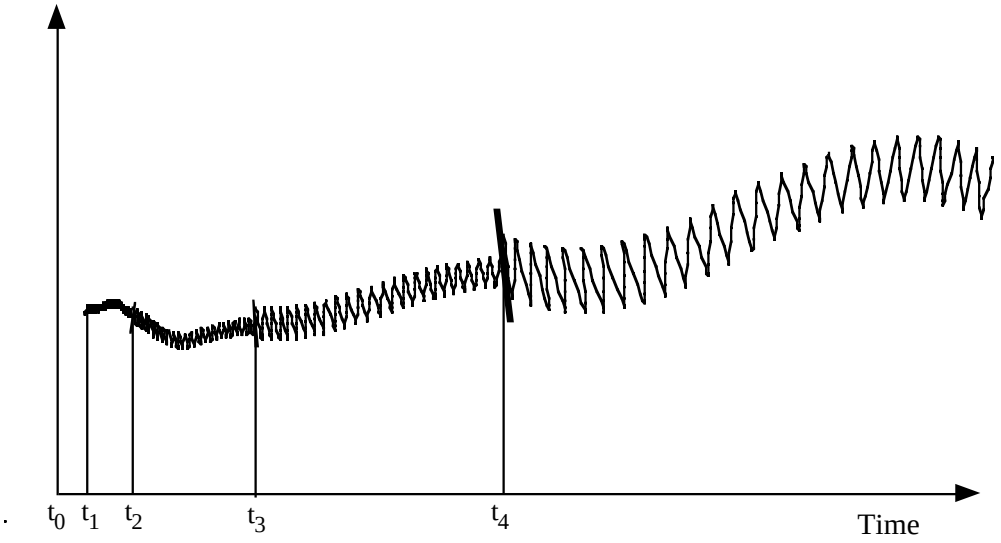


Figure 4: Basic Schema for multi-resolutional system response.

The ÒslinkyÓ for interval $[t_3, t_4)$ interfaces with a larger Òslinky,Ówith less resolution, at t_4. It also interfaces with a shorter Òslinky,Ówith greater resolution at t_3. Assume also that each ÒslinkyÓ can manage its shape. However, no ÒslinkyÓ can act independently of the others. Specifically, the Òslinky Òon interval $[t_3, t_4)$ must interact with the ÒslinkyÓ operating beyond t_4 in order to define the boundary conditions at t_4. To be more precise, the Task Accepting function of the $[t_3, t_4)$-subsystem must interact with the Tasking Assigning functions for the Assignors of the $[t_3, t_4)$-subsystem. Similarly, the Task Assigning function for the $[t_3, t_4)$-subsystem must interact with the Task Accepting functions of its Acceptors. Finally, the shape of the ÒslinkyÓ between t_3 and t_4 is controlled by the Performance Improvement function as the $[t_3, t_4)$-subsystem responds to forecasted external inputs that will likely act upon it during the interval $[t_3, t_4)$.

Given the recursive system-of-systems nature for the system, each Acceptor for the $[t_3, t_4)$-subsystem interacts with the $[t_3, t_4)$-subsystem in a manner similar to the way that the $[t_3, t_4)$-subsystem interacts with its Assignors. Similarly, each Acceptor for the $[t_3, t_4)$-subsystem will similarly interface with its own Acceptor=s Task Accepting functions. Note also that none of the indicated subsystems can touch t_0 because only a Process that has no Acceptors can reach t_0. (We will discuss this assertion later.)

Now, let us try to visualize the dynamic behavior of the proposed system=s response. Remember, t_0 (the current time) must continue to advance in real time. We may assume that the entire state trajectory is dynamic, and neither the interface times (t_1, t_2, É) nor the boundary conditions are fixed. Instead, the shared boundary conditions between two adjacent ÒslinkiesÓ are constantly being renegotiated in real time. As the boundary conditions are modified and the forecasts for the external effects upon a given subsystem are updated, the intelligent controller responds by modifying the projected desired shape of the ÒslinkyÓ between the appropriate interface times.

We have stated that only Processes can affect the system in real time. Several important conclusions follow:

- *No interface time at the junction of two subsystems' responses can ever occur*. These interface times constantly change with time and must always be greater than the current time t_0.
- *Only Processes react to real inputs from the external environment*. The other subsystems plan their response based upon forecasted inputs from the environment and their current negotiated boundary conditions.
- *The planned trajectories of the non-processing subsystems are never realized*. These planned trajectories only conjecture how the system will likely respond for planning purposes.
- *The purpose of the intelligent controllers for the non-processing subsystems is simply to establish goals for another subsystem*. The recursive system-of-system nature of these systems implies that these goals will become more detailed as their interfacing times approach t_0.

Figure 4 does not adequately depict the interaction between a given subsystem and its Assignor(s) and Acceptors. In Figure 5, we provide a more detailed illustration of the proposed interaction among the subsystems as they interact with each other. It also illustrates the evolution of time and the limitations that a given system has in managing the response of the system.

Time advances from left to right in Figure 5. The large sphere represents the state space for the aggregate subsystem that projects into the most distant future. Within that subsystem=s state space, there are two smaller spheres. The right-most of these spheres represents the goal space that the system seeks to reach at t_4. In this case, we assume that the final goal is established by the system=s environment because the Assignors for this subsystem have not been included within the system model. Note that this is an arbitrary choice based upon the modeler=s desires and is determined to a certain extent by how far the modeler wants to forecast the system=s future response.

The left-most sphere within the largest sphere represents the forecasted state at t_3 from which the aggregate subsystem initiates its planning. Thus, the aggregate subsystem represented by the largest sphere will plan on the interval $[t_3, t_4)$. The values for both t_3 and t_4 are dynamic and must increase with real-time, and t_4 is always greater than t_3. (The reader will note that we have not included t_4 within the subsystem=s planning horizon because the goal state at t_4 is specified by an agent outside of the modeled system).

The role of the intelligent controller for the $[t_3, t_4)$-subsystem is to determine the ideal trajectory from the anticipated state at t_3 to the desired goal state at t_4. During that interval, the subsystem must also respond to other external inputs. Because the planning interval is beyond the current time, these external inputs must be forecasted. Hence, the planned response on the $[t_3, t_4)$ interval is a projected response only. It will not (or cannot) be implemented as planned.

The $[t_3\text{-}t_4)$-subsystem cannot manage the response of the system before t_3 because it cannot address the detail required to describe the system's response prior to t_3. Rather, this detail will be addressed by two other subsystems as indicated by the second largest spheres in Figure 5. The fact that the spheres are smaller has no relation to the dimensions of each subsystem's state space. Rather, the diameters of the spheres correspond to the relative length of the planning interval that each subsystem addresses.

The $[t_3, t_4)$ subsystem estimates its initial state at t_3. Thus state is achieved by the subordinate's response on the $[t_2, t_3)$ time interval. In order to manage the subordinate subsystem's response, the $[t_3, t_4)$-subsystem must define goal states for the two subsystems at $t_3^{(1)}$ and $t_3^{(2)}$, respectively. However, the state variables considered by the subordinate subsystems are different than those considered by the $[t_3, t_4)$-subsystem. Hence, a transformation between the state spaces must occur. This transformation is implemented by the Task Assignor for the $[t_3, t_4)$-subsystem as it interacts with the Task Acceptors within $[t_3,t_4)$-subsystem's Acceptors. This transformation involves two types of interactions. With respect to the $[t_3, t_4)$-subsystem, the first interaction determines a mutually acceptable set of feasible goals for each Acceptor. The second interaction monitors each Acceptor's progress in achieving its assigned goals. Here, the Task Assigning function for $[t_3, t_4)$-subsystem must transform each Acceptor's projected goal achievement into the corresponding state representation within the $[t_3, t_4)$-subsystem's state space. Moreover, the individual Acceptor's response must be integrated to form a single composite estimate for the $[t_3, t_4)$-subsystem's initial state at t_3.

The goals established for the Acceptors will cover the time interval up to $t_3^{(1)}$ and $t_3^{(2)}$, respectively. In order to insure planning across the entire time interval up to t_4, it is essential that t_3 be less than or equal to either $t_3^{(1)}$ or $t_3^{(2)}$. Thus, the planning interval for a given Acceptor usually overlaps the planning interval of its Assignor(s). In addition, the state space for the individual Acceptors can also overlap each other. For example, it might be possible for both Acceptors to execute a given task. It is also possible that the state spaces are not entirely congruent. One Acceptor might be able to execute tasks that the other Acceptor cannot.

On the other hand, the state trajectories through the subsystem's state spaces must not intersect. Two distinct subsystems may not perform identical tasks upon the same entity at the same time. Two or more subsystems could possibly collaborate, but one subsystem would still assume primary control of the entity and subsystem actions upon the entity must differ from the others in some

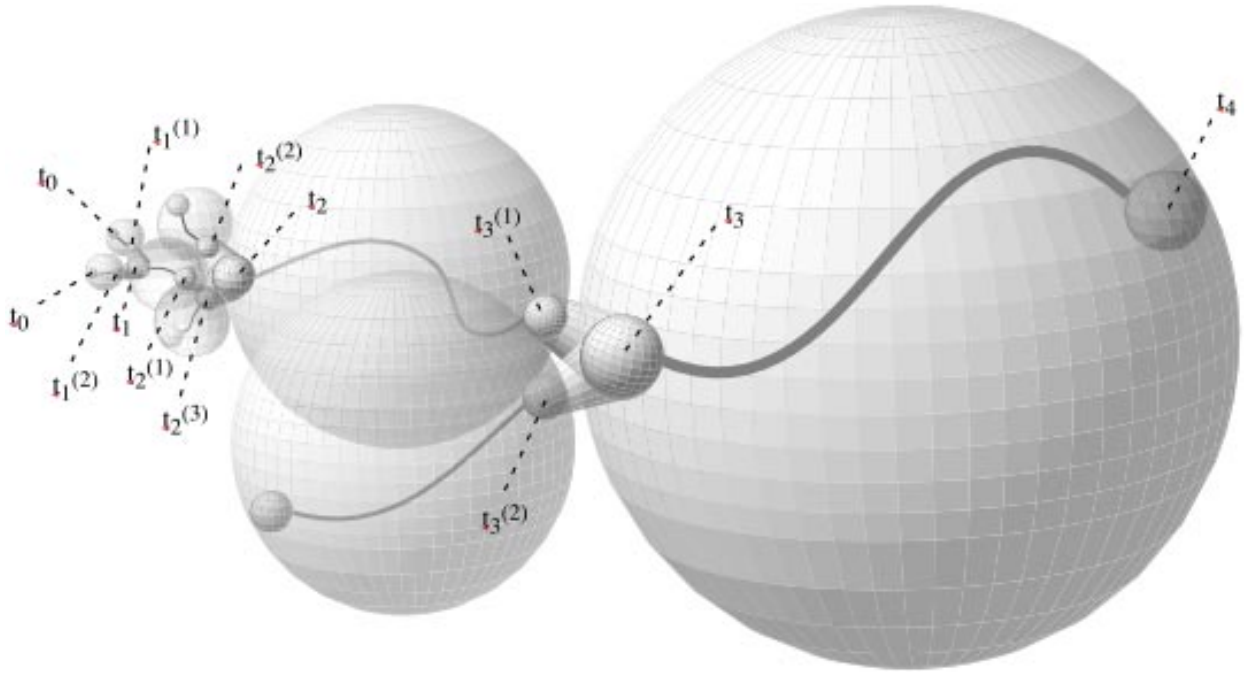


Figure 5. A more detailed representation of the multi-resolutional state evolution.

manner at a given time. A fundamental law of physics prevents two objects from occupying the same region of space and time simultaneously.

Given its desired final goal state at $t_3^{(1)}$, the $[t_2, t_3^{(1)})$-subsystem plans its response through its state space. The $[t_2, t_3^{(1)})$-subsystem interacts with its Acceptors in order to estimate its initial planning state at t_2. This interaction also establishes the goals for each of the $[t_2, t_3^{(1)})$-subsystem's Acceptors at $t_2^{(1)}$, $t_2^{(2)}$ and $t_2^{(3)}$. Having established each of their individual goals, the $[t_2, t_3^{(1)})$-subsystem's Acceptors can determine their individual initial planning states. The same process is repeated for each state trajectory emanating from t_4 until a process, (which has no Acceptors), is encountered. This terminating process can be managed at t_0. Hence, the recursive planning process generates a collection of state trajectories, each beginning at t_0 and terminating at t_4.

Figure 5 depicts a situation where hierarchical planning occurs. Each subsystem has at most one Assignor, and the collection of state trajectories illustrated in Figure 5 forms a tree. Observe that state trajectories continue to divide as the diagram progresses from the most distant time t_4 toward present time. Moreover, each subsystem has a single state trajectory to manage. In Figure 5, we did not include every possible subsystem in order to simplify the figure. (Observe that some of the state trajectories do not begin at t_0.) If all potential subsystems for a hierarchical system were included, then every path in the state-trajectory tree would start at t_0 and terminate at a common root at t_4.

Recently, hierarchical systems have fallen from favor. Certainly, hierarchies, like all organizational structures, do have their limitations. However, most limitations occur when the Assignors dictate their goal assignments and the Acceptors cannot reject an assigned goal. Recent management and distributed planning approaches seek to empower the subordinate subsystems with greater planning and control responsibilities. Contrary to popular belief, such empowerment does not negate hierarchical structures.

Unfortunately, there are situations where hierarchies are inappropriate. For example, the government typically seeks to prevent individual corporations from collaborating in order to create a monopolistic environment.

One benefit of the proposed approach is that we can now characterize the limitations that arise when one must employ a structure other than a hierarchy. In particular, we can now test many of the claims that the advocates of other architectures have cited.

5. Conclusions

The efficacy of current planning and control technologies requires a monolithic statement of the system's planning and control problems. If such monolithic statements cannot be made, then available planning and control technologies are probably irrelevant. The above discussion demonstrates that it will be impossible to provide such monolithic specifications for most complex systems.

One might question whether it is possible to provide a monolithic statement for any system's planning and control problems. Remember that one's definition of the system's boundary is arbitrary. In most cases, the defined system is still dependent upon other environmental subsystems that are being managed by other entities. A subsystem seldom has complete control over its planning and control responsibilities. Witout such control, it is impossible to demonstrate optimality of a planned /executed system response with respect to any performance criteria. Given the current state of affairs, performance evaluations for a given subsystem level or the composite system should be avoided. The primary goal must be to develop improved technologies for managing complex systems.

6. References

[1] L. S. Lasdon, Optimization Theory for Large systems, London, Macmillian Company, 1970.

[2] D. A. Wismer, Optimation Methods for Large-Scale Systems with Applications, New York, MCGraw-Hill Book Company, 1971.

[3] P. Kokotovic, H. Khalil and J. O'reilly, Singular Perturbation Methods in Control: Analysis and Design, New York, Academic Press, 1986.

Hypothesis Testing for Complex Agents

Joanna Bryson, Will Lowe† and Lynn Andrea Stein
MIT AI Lab
Cambridge, MA 02139
joanna@ai.mit.edu, las@ai.mit.edu

†Tufts University Center for Cognitive Studies
Medford, MA 02155
wlowe02@tufts.edu

Abstract

As agents approach animal-like complexity, evaluating them becomes as difficult as evaluating animals. This paper describes the application of techniques for characterizing animal behavior to the evaluation of complex agents. We describe the conditions that lead to the behavioral variability that requires experimental methods. We then review the state of the art in psychological experimental design and analysis, and show its application to complex agents. We also discuss a specific methodological concern of agent research: how the robots versus simulations debate interacts with statistical evaluation. Finally, we make a specific proposal for facilitating the use of scientific method. We propose the creation of a web site that functions as a repository for platforms suitable for statistical testing, for results determined on those platforms, and for the agents that have generated those results.

Keywords: *agent performance, complex systems, behavioral indeterminacy, replicability, experimental design, subjective metrics, benchmarks, simulations, reliability.*

1. Introduction

Humanoid intelligence is a complex skill, with many interacting components and concerns. Unless they are in an exceptional, highly constrained situation, intelligent agents can never be certain they are expressing the best possible behavior for the current circumstance. This is because the problem of choosing an ordering of actions is combinatorially explosive [9]. Consequently, for scientists or engineers evaluating the behavior of an agent, it is generally impossible to ascertain whether a behavior is optimal for that agent. Albus [2] defines intelligence as "the ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success." Systems of such complexity are rarely amenable to proof-theoretic techniques [26]. In general, the only means to judge an increase in probability is to run statistical tests over an appropriately sized sample of the agent's behavior.

Computational systems, in contrast, are traditionally evaluated based on their *final* results and/or on their resource utilization [29]. The historical definition of computational process (c.f. Babbage, Turing, von Neumann) is modeled on mathematical calculation, and its validity is measured in terms of its ultimate product. If the output is correct — if the correct value is calculated — then the computation is deemed correct as well. More recent descriptions [e.g. 11] have added an assessment of the time, space, processor, and other resource utilization, so that a computation is only deemed correct if it calculates the appropriate value within some resource constraints.

This characterization of computation is less applicable when it comes to particular operating systems and other real-time computational systems. These systems have no final result, no end point summarizing their work. Instead, they must be evaluated in terms of ongoing behavior. Guarantees, where they exist, take the form of performance constraints and temporal invariants. Although formal analysis of correctness plays a role even in these systems, performance testing, including benchmarking, is an essential part of the evaluation criteria for this kind of computational system.

Computational agent design owes much to computer science. But the computationalist's tendency to evaluate in terms of ultimate product is as inappropriate for computational agents as it is for operating systems. Instead, metrics must be devised in terms of ongoing behavior, performance rather than finitary result. But what is the analog to benchmarking when the tasks are under-specified, ill-defined, and subject to interpretation and observer judgment?

In this paper, we will examine issues of running such evaluations for complex agents. By *complex agents* we mean autonomous agents such as robots or VR characters capable of emulating humanoid or at least vertebrate intelligence. We will discuss hypothesis testing, including the statistical controversies that have lead to the recent revisions in the standard experi-

mental analysis endorsed by the American Psychological Association. We will also discuss recent advances in methodologies for establishing quantitative metrics for matters of human judgment, such as whether one sentence is more or less grammatical than another, or an anecdote is more or less appropriate. We propose a means to facilitate hypothesis testing between groups: a simulation server running a number of benchmark tests.

2. Motivation: Sources of Uncertainty

Although there is certainly a role for using formal methods in comparing agent architectures [e.g. 8, 6], what we as agent designers are ultimately interested in is comparing the resulting *behavior* of our agents. Given the numerous complex sources of indeterminacy in this behavior, such comparison requires the application of the same kind of experimental methodology that has been developed by psychology to address similar problems. In this section we review some of the sources of this indeterminacy; in the next we will review analytic approaches for addressing them.

The first source of indeterminacy is described above: The combinatorial complexity of most decision problems makes absolute optimality an impractical target. Thus even if there is a single unique optimal sequence of actions, in most situations we cannot expect an agent to find it. Consequently, we will expect a range of agents to have a range of suboptimal behaviors, and must find a way of comparing these.

The next source of indeterminacy is the environment. Many agents must attempt to maintain or achieve multiple, possibly even contradictory goals. These goals are often themselves uncertain. For example, the difficulty of eating is dependent on the supply of food, which may in turn be dependent on situations unknowable to the agent, whether these be weather patterns, the presence or absence of other competing agents, or in human societies, local holidays disrupting normal shopping. Thus in evaluating the general efficacy of an agent's behavior, we would need a large number of samples across a range of environmental circumstances.

Another possible source of indeterminacy is the development of agents. As engineers, we are not really interested in evaluating a single agent, but rather in improving the state-of-the-art in agent design. In this case, we are really interested in what approaches are most likely to produce successful agents. This involves uncertainty across development efforts, complicated by individual differences between developers. Many results contending the superiority or optimality of a particular theory of intelligence may simply reflect effective design by the practitioners of that theory [e.g. 7].

Finally, the emphasis of this workshop is on natural, human-like behavior. Humans are highly social animals, and social acceptability is an important criteria for intelligent agents. However, sociability is not a binary attribute: it varies in degrees. Further, a single form of behavior may be considered more or less social by the criteria of various societies. Evaluations of systems by such criteria requires measurement over a population of judges.

3. Current Approaches to Hypothesis Testing

The previous section presented a number of challenges to the evaluation of complex, humanoid agent building techniques. In this section we review methodologies used by psychology — the evaluation of human agents — that are available to address these challenges.

Although it is obvious that comparing two systems requires testing, the less obvious issues are how many tests need to be run and what statistical analysis needs to be used in order to answer these questions. In this section we describe three increasingly common problems in Artificial Intelligence and discuss a set of experimental techniques from the behavioral sciences that can be used to address them.

The first problem is variability in results: We need to know whether performance differences that arise over test replications can be ascribed to varying levels of a system's ability or to variation in lighting conditions, choice of training data, starting position, or some other or some other external (and therefore uninteresting) source. Psychology uses statistical techniques such as the Analysis of Variance (ANOVA) to address these issues. The second problem is of disentangling complex and unexpected interactions between subparts of a complex system. This can also be addressed using ANOVA coupled with factorial experimental design. The third problem is that of rigorously and meaningfully evaluating inherently subjective data. Since many psychology experiments investigate inherently subjective matters, the field has developed a set of techniques that will be of use to artificial agent designers as well. The next three sections describe these solutions in more detail.

3.1 Variability in Results

The problem of comparing performance variability due to differences in ability and variability due to extraneous factors is ubiquitous in psychology. It is dealt with by procedures known collectively as Analysis of Variance or ANOVA.

3.1.1 Standard ANOVA

In a typical experimental design for comparing performance, K systems are tested N times each. If the variation in performance between the K systems outweighs the variability among each system's N runs, then the system performances are said to be *significantly different*. We then examine the systems pairwise to get information about ordering. The ANOVA allows us to infer that e.g. although there are differences overall between the K=4 systems (i.e. some are better than others), the performance difference between 3 and 4 is reliable, whereas the difference between 1 and 2 is not reliable because it is outweighed by the amount of extraneous variation across the N tests. In this case, although 1 may perform on average better than 2, this does not

imply that it is actually better on the task. If the experiment were repeated then 2 would have reasonable chance of performing on average the same as 1, or even better.

The notion of *reasonable chance* used above is the essence of the concept of significant difference. System 3 is on average better than 4 in this experiment and the ANOVA tells us that performances are significantly different at the .05 level (expressed as $p<.05$). This means that in an infinite series of experimental replications, if 3 is in fact exactly *as good* as 4, i.e. there is no genuine performance difference, then the probability of getting a performance difference as large or larger than the one observed in this experiment is 0.05. The smaller this probability becomes, the more reliable the difference is. In contrast, the fact that the average performances of 1 and 2 are not significantly different means their ordering in this experiment is not reliable because there is a more than 0.05 probability that the ordering would not be preserved in a replication.

Notice that hypothesis testing using ANOVA does not *guarantee* an ordering, it presents probabilities that each part of the ordering is reliable. This is a fundamental difference between experimental evidence and proof. Scientific method increases the probability that hypotheses are correct but it does not demonstrate them with complete certainty.

The binary output of hypothesis tests (significant difference versus no significant difference) and its probability is an unnecessarily large loss of information. The American Psychological Association have consequently recently moved to emphasize confidence intervals over simple hypothesis testing. A *confidence interval* is a range, centered on the observed difference, that in the hypothetical replications will contain the true performance value some large percentage, say 95%, of the time. In the example above, each system has a 95% confidence interval, or *error bar*, centered on its average performance with width determined by the amount of variability between runs. When two intervals overlap, there is a significant probability that a replication will not preserve the current ordering among the averages and we can conclude that the corresponding performance difference is unreliable. This method gives the same result as simple hypothesis testing above — the performances are not significantly different — but is much more informative: confidence intervals give an idea about how much variability there is in the data itself and yield a useful graphical representation of analytical results.

3.1.2 Alternative Approaches to Analysis

Stating confidence intervals is more informative than simple significance judgments. However, it also relies on an hypothetical infinity of replications of an experiment. This aspect of classical statistical inference is a result of assuming that the true difference in performance is fixed and the observed data is a random quantity. Alternatively, in Bayesian analysis the difference is considered uncertain and is modeled as a random variable whereas the results are fixed because they have already been observed [5]. The result is a probability distribution over values of the true difference. To summarize the distribution an interval containing 95% of the probability mass can be quoted. This takes the same form as a confidence interval, except that its interpretation is much simpler: Given the observed results, the probability that the true difference is in the interval is 0.95, so if the interval contains 0, there is a high probability that there is no real performance difference between systems.

The Bayesian approach makes no use of hypothetical experimental replications and is more naturally extended to deal with complicated experimental designs. On the other hand, it does require an initial estimate (or prior distribution) for the probabilities of various values of the performance difference before seeing test data. There is much controversy about which of these approaches is more appropriate. In the context of AI however, we need not take a stand on this issue. The two approaches answer different questions, and for our purposes the questions answered by classical statistics are of considerable interest. Unlike many of the natural sciences, the performance of AI systems over multiple replications is not only accessible, but of particular interest. To the extent we are engineers, AI researchers must be interested in reliability and replicability of results.

3.2 Testing for Interacting Components

Many unpleasant software surprises arise from unexpected interactions between components. Unfortunately, in a complex system it is typically infeasible to discover the nature of interactions analytically in advance. Consequently *factorial experimental design* is an important empirical tool.

As an example, assume that we can make two changes A and B to a system. We could compare the performance of the system with A to the same system without it, using the ANOVA methods above, and then do the same for B. But when building a complex system it is essential to also know how A and B affect performance together. Separate testing will never reveal, for example, that adding A generates a performance improvement only when B is present and not otherwise. This is referred to as an *interaction* between A and B, and can be dealt with by testing all combinations of system additions, leading to a factorial experiment. Factorial experiments are analyzed using simple extensions to ANOVA that test for significant interactions as well as simple performance differences. Factorial ANOVA methods are described in any introductory statistics textbook [e.g 23].

In the discussion above we have implicitly assumed that differences in performance can be modeled as continuous quantities, such as distance traveled, length of conversation or number of correct answers. When the final performance measure is discrete, e.g. success or failure, then *logistic regression* [1, ch.4] is a useful way to examine the effects of additions or manipulations on the system's success rate. Information about the effects of arbitrary numbers of additions, both individually and in in-

teraction, is available using this method, just as in the factorial ANOVA. Logistic regression also gives a quantitative estimate of *how much* the probability of success changes with various additions to the system, which gives an idea of the importance of each change.

3.3 Quantifying Inherently Subjective Data

Often performance evaluation involves judgments or ratings from human subjects. Clearly it is not enough that one subject judges an AI conversation to be lifelike because we do not know how typical that subject is, and how robust their opinion is. It would be better to choose a larger sample of raters, and to check that their judgments are reliable. When ratings are discrete (good, bad) or ordinal (terrible, bad, ok, good, excellent) then Kappa [22] is a measure of between-rater agreement that varies from 1 (perfect agreement) to -1 (chance levels of agreement). For judgments of continuous quantities the intraclass correlation coefficient [13] performs the same task.

However, such discrete classifications are often clumsy. Because a rating system is itself subjective, the extra variance added by difference in interpretation of a category can lose correlations between subjects that actually agree on the relative validity or likeability of two systems. Further, we would really prefer in many circumstances to have a continuous range of difference values. Such results can be provided by *magnitude estimation*, a technique from psychophysics. For example, Bard *et al.* [4] have recently introduced the use of magnitude estimation to allow subjects to judge the acceptability of sentences which have varying degrees of syntactic propriety. In a magnitude estimation task, each subject is asked to assign an arbitrary number as a value for the first example they see. For each subsequent example, the subject need only say how much more or less acceptable it is, with reference to the previous value, e.g. twice as acceptable, half as acceptable and so on. This allows subjects to pick a scale they feel comfortable with manipulating, yet gives the experimenter a generally useful metric. For example, in Bard *et al.*'s work, a subject might give the first sentence an 8, the next a 4, the following a 32 — the experimenter records 1s, .5s and 4s respectively. This method has been shown to reduce the number of judgments necessary to get very reliable and accurate estimates of acceptability, relative to other methods.

Bard *et al.* manipulate the sentences themselves, but it is clear that magnitude estimation can equally well be used to get fine-grained judgments about how natural the output of a natural language processing (NLP) system is, and the degree to which this is improved by adding new components. Nor is the method limited to linguistic judgments, for it should be equally effective for evaluating ease of use for teaching software, the psychological realism of virtual agents or the comprehensibility of output for theorem proving machinery.

4. Environments for Hypothesis Testing: Robots and Simulations

As the previous sections indicate, one of the main attributes of statistically valid comparisons is a large number of experimental trials. Further, these experimental conditions should be easily replicable and extendible by other laboratories. In Section 5. we propose that a good way to facilitate such research is to create a web location dedicated to providing source code and statistics for comparative evaluations over a number of different benchmark tasks. This has approach has proven useful in neural network research, and should also be useful for complex agents. However, it flies in the face of one of the best-known hypotheses of complex agent research: that good experimental method requires the use of robots. Consequently, we will first provide an updated examination of this claim.

4.1 Arguments Against Simulation

Simulation is an attractive research environment because it is easy to maintain valid controls, and to execute large numbers of replications across a number of machines. However, there have been a number of important criticisms leveled against this approach.

A Simulations never replicate the full complexity of the real world. In choosing how to build a simulation, the researcher first determines the 'real' nature of the problem to be solved. Of course, the precise nature of a problem largely determines its solution. Consequently, simulations are not valid for truly complex agents, because they do not test the complete range of problems a natural or embodied agent would face.

B If a simulation truly were to be as complicated as the real world, then building it would cost more time and effort than can be managed. It is cheaper and more efficient to build a robot, and allow it to interact with the real world. This argument assumes one of basic hypotheses of the behavior-based approach to AI [3], that intelligence is by its nature simple and its apparent complexity only reflects the complexity of the world it reacts to. Consequently, spending resources constructing the more complicated side of the system is both irrational and unlikely to be successful.

C When researchers build their own simulations, they may deceive either themselves or others as to the validity or complexity of the agents that operate in it. Since both the problem and the solution are under control of the researcher, it is difficult to be certain that neither unconscious nor deliberate bias has entered into the experiments. In contrast, a robot is considered to be clear demonstrations of autonomous artifact; its achievements cannot be doubted, because it inhabits the same problem space we do.

4.2 Are Robots Better than Simulations?

These arguments have led to the wide-spread adoption of the autonomous robot as a research platform, despite the known problems with the platform [16]. These problems reduce essentially to the fact that robots are extremely costly. Although their popularity has funded enough research and mass production to reduce the initial cost of purchase or construction, they are still relatively expensive in terms of researcher or technician time for programming, maintenance, and experimental procedures. This has not prevented some researchers from conducting rigorous experimental work on robot platforms [see e.g. 10, 25]. However, the difficulty of such procedures adds urgency to the question of the validity of experiments in simulation.

This difficulty has been reduced somewhat by the advent of smaller, more robust, and cheaper mass-produced robot platforms. However, these platforms still fall prey to a second problem: mobile robots do not necessarily address the criticisms leveled above against simulations better than simulations do. There are two reasons for this: the need for simplicity and reliability in robots, and the growing sophistication of simulations.

The constraints of finance, technological expertise and researchers' time combine to make it extremely unlikely that a robot will operate either with perception anything near as rich as that of a real animal, nor with actuation having anything like the flexibility or precision of even the simplest animals. Meanwhile, the problem of designing simulations with predictive value for robot performance has been recognized and addressed as a research issue [e.g. 18]. All major research robot manufacturers now distribute simulators with their hardware. In the case of Khepera, the robot most used by researchers running experiments requiring large numbers of trials, the pressure to provide an acceptable simulator seems to have not only resulted in an improved simulator, but also a simplified robot, thus making results on the two platforms nearly identical. Clearly this similarity of results either validates the use of the Khepera simulator, or invalidates the use of the robot.

When a simulator is produced independent of any particular theory of AI as a general test platform, it defeats much of the objection raised in charges *A* and *C* above, that a simulator is biased towards a particular problem, or providing a particular set of results. In fact, complaint *C* is particularly invalid as a reason to prefer robotics. Experimental results provided on simulations can be replicated precisely in other laboratories. Consequently, they are generally *more easily* tested and confirmed than those collected on robots. To the extent that a simulation is created for and possibly by a community — as a single effort resulting in a platform for unlimited numbers of experiments by laboratories world-wide, that simulation also has some hope of overcoming argument *B*.

This gross increase in the complexity of simulations has particularly true of two platforms. First, the simulator developed for the simulation league in the RoboCup soccer competition has proven enormously successful. Although competition also takes place on robots, to date the simulator league provides far more "realistic" soccer games in terms of allowing the demonstration of teamwork between the players and flexible offensive and defensive strategies [21, 19]. This success has encouraged the RoboCup organization to tackle an even more complex simulator designed to replicate catastrophic disasters in urban settings [20]. This simulator is intended to be sufficiently realistic as to eventually allow for swapping in real-time sensory data from disaster situations, in order to allow disaster relief to monitor and coordinate both human and robotic rescue efforts.

The second platform is also independently motivated to provide the full complexity of the real world. This is the commercial arena of virtual reality (VR), which provides a simulated environment with very practical and demanding constraints which cannot easily be overlooked. Users of virtual reality bring expectations from ordinary life to the system, and any agent in the system is harshly criticized when it fails to provide adequately realistic behavior. Thórisson [30] demonstrates that users evaluate a humanoid avatar with which they have held a conversation as much more intelligent if it provides back-channel feedback, such as eyebrow flashes and hand gestures, than when it simply generates and interprets language. Similarly Sengers [27] reviews evidence that users cannot become engaged by VR creatures operating with overly reactive architectures, because the agents do not spend sufficient time telegraphing their intentions or deliberations. Such constraints have often been overlooked in robotics.

In contrast, robots which must be supported in a single lab with limited technical resources are likely to deal with far simpler tasks. Robots may face far fewer conflicting goals, lower time-related conflicts or expectations, and even fewer options for actuation. Although robots still tend to have more natural perceptual problems than simulated or VR agents, even these are now increasingly being addressed with reliable but unnatural sensors such as laser range finders.

4.3 Roles for Robots and Simulations

Robots are still a highly desirable research platform. They provide complete systems, requiring the integration of many forms of intelligence. Many of the problems they need to solve are closely related to animal's problems, such as perception and navigation. In virtual reality, perfect perception is normally provided, but motion often has added complication over that in the real world. Depending on the quality of the individual virtual reality platform, an agent may have to deliberately not pass through other objects or to intentionally behave as if it were affected by gravity or air resistance. Even in the constantly improving RoboCup soccer simulator, there are outstanding difficulties in simulating important parts of the game, such as the goalkeeper's ability to kick over opposing team members (currently compensated for by allowing the keeper to "warp" to any point in the goal box instantaneously when already holding the ball.)

Robots being embodied in the real world are still probably the best way to enforce certain forms of honesty on a researcher. A mistake cannot be recovered from if it damages the robot, an action once executed cannot be revoked. Though this is also true of some simulations [e.g. 31], particularly in the case of younger students, these constraints are better brought home on a robot, as it becomes more apparent why one can't 'cheat.' Finally, building intelligent robots is a valid end in itself. Commercial intelligent robots are beginning to prove very useful in care-taking and entertainment, and may soon prove useful in areas such as construction and agriculture. In the meantime robots are highly useful in the laboratory for stirring interest and enthusiasm in students, the press and funding agencies. However, given the arguments above, we conclude that the use of robots as experimental platforms is neither necessary nor sufficient in providing evidence about complex agent intelligence. Robots, like simulations, must be used in combination with rigorous experimental technique, and even so can only provide evidence, not conclusive proof, of agent hypotheses.

In summary, neither robots nor simulation can provide a single, ultimate research platform. But then, neither can any other single research platform or strategy [15]. While not denying that intelligence is often highly situated and specialized [14, 17], to make a general claim about agent methodology requires a wide diversity of tasks. Preference in platforms should be given to those on which multiple competing hypotheses can be tested and evaluated, whether by qualitative judgments such as the preference of a large number of users, or by discrete quantifiable goals to be met, such as a genetic fitness function, or the score of a soccer game.

5. Coordinating Hypothesis Testing

Whether there can be general solutions to problems of intelligence is an empirical matter that has already been tested in some domains. For neural networks and other machine learning methods, the UCI Machine Learning Repository holds a large collection of benchmark learning tasks. Besting these benchmarks is not a necessary requirement for the publication of a new algorithm, but showing a respectable performance on them improves the reception of new contributions. Essentially, benchmarks are one indication for both researchers and reviewers of when an innovation is likely to be of interest.

Further, Neal and colleagues at the University of Toronto have constructed DELVE [24], a unified software framework for benchmarking machine learning methods. DELVE contains a large number of benchmark data sets, details of various machine learning techniques, currently mostly neural networks and Gaussian Processes, and statistical summaries of their performance on each task. One of the most important requirements is that each method is described in enough detail that it could be implemented by another researcher and would obtain a similar performance on the tasks. This ensures that the mundane but essential decisions that are an essential part of many learning algorithms (e.g. setting weight decay parameters, choosing k in k-nearest-neighbor rules) are not lost.

We propose a complex agent comparison server or web site, to be at least partially modeled on DELVE. This site should allow for the rating of both agent approaches and comparison environments, thus encouraging and facilitating research in both fields. It could also be annotated for educational purposes, indicating challenges and environments well suited to school, undergraduate, and graduate course projects. Such a site might provide multiple indices, such as:

- Environments, ranked by number and/or diversity of participants.
- Agent architectures (e.g. Soar, Behavior-Based AI). This should also allow for the petition for new categories.
- Contestants and/or contesting labs or research groups . This allows researchers interested in a particular approach to see any related work. Ranked by the number and/or diversity of environments.

Here are some examples of already existent platforms which might be included on the server:

- RoboCup [21, 19].
- Khepera robot competitions. Both of these two suggestions provide simulations as well as organized robotic competitions. They test learning and perception as well as planning or action selection.
- Tile World and Truck World, designed as complex planning domains. [15]
- Tyrrell's Simulated Environment [31] designed to test action-selection and goal management.
- Chess.
- An analog Turing Test, using magnitude estimation to compare dialog systems.

In addition, there are at least two software environments designed specifically to allow testing and comparison of a number of different architectures, though they contain no specific experimental situations as currently developed. These environments are Cogent [12] and the Sim_agent Toolkit [28].

6. Conclusion

To summarize, we believe that as agents approach the goal of being psychologically realistic and relevant, their evaluation will require the techniques that have been developed in the psychological sciences. This evaluation is critical in providing a gradient as we search for the right sorts of techniques to build complex agents. The techniques of hypothesis testing have been refined to describe truly complex agents. However, these are scientific techniques, not proofs. They do not give us certain

answers, only more information. We believe many of the criticisms of benchmark testing made in the past failed to properly acknowledge this feature of experimentation. We should trust increased probability, rather than proof-theoretic guarantees. The more people perform tests across competing hypotheses, the more likely we will be to achieve our research goals, whether they are engineering complex, social agents, or understanding the nature of intelligence.

Acknowledgments

The authors would like to acknowledge early discussions with Brendan McGonigle and Ulrich Nehmzow on this topic.

References

[1] A. Agresti. *Categorical Data Analysis*. John Wiley and Sons, 1990.

[2] J. S. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):473–509, 1991.

[3] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.

[4] E. Bard, D. Robertson, and A. Sorace. Magnitude estimation of linguistic acceptability. *Language*, 72(1):32–68, 1996.

[5] G. E. P. Box and G. C. Tiao. *Bayesian inference in statistical analysis*. Addison-Wesley, Reading, Massachusetts, 1993.

[6] Joanna Bryson. Cross-paradigm analysis of autonomous agent architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):165–190, 2000.

[7] Joanna Bryson. Hierarchy and sequence vs. full parallelism in reactive action selection architectures. In *From Animals to Animats 6 (SAB00)*. MIT Press, 2000.

[8] Joanna Bryson and Lynn Andrea Stein. Architectures and idioms: Making progress in agent design. In *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL2000)*, 2000. to be presented July 2000.

[9] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[10] David Cliff, Philip Husbands, and Inman Harvey. Explorations in evolutionary robotics. *Adaptive Behavior*, 2(1):71–108, 1993.

[11] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on the THeory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery.

[12] R. Cooper, P. Yule, J. Fox, and D. Sutton. COGENT: An environment for the development of cognitive models. In U. Schmid, J. F. Krems, and F. Wysotzki, editors, *A Cognitive Science Approach to Reasoning, Learning and Discovery*, pages 55–82. Pabst Science Publishers, Lengerich, Germany, 1998. see also http://cogent.psyc.bbk.ac.uk/.

[13] P. E. Fleiss and J. L. Shrout. Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin*, 86(2):420–428, 1979.

[14] C.R. Gallistel, Ann L. Brown, Susan Carey, Rochel Gelman, and Frank C. Keil. Lessons from animal learning for the study of cognitive development. In Susan Carey and Rochel Gelman, editors, *The Epigenesis of Mind*, pages 3–36. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[15] Steve Hanks, Martha E. Pollack, and Paul R. Cohen. Benchmarks, testbeds, controlled experimentation and the design of agent architectures. Technical Report 93–06–05, Department of Computer Science and Engineering, University of Washington, 1993.

[16] Ian D. Horswill. *Specialization of Perceptual Processes*. PhD thesis, MIT, Department of EECS, Cambridge, MA, May 1993.

[17] Ian D. Horswill. *Specialization of Perceptual Processes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1993.

[18] N. Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. *Journal Of Adaptive Behaviour*, 6(2):325–368, 1997.

[19] Hiroaki Kitano. Special issue: Robocup. *Applied Artificial Intelligence*, 12(2–3), 1998.

[20] Hiroaki Kitano. Robocup rescue: A grand challenge for multiagent systems. In *The Fourth International Conference on MultiAgent Systems (ICMAS00)*, pages 5–12, Boston, 2000. IEEE Computer Society.

[21] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In *Proceedings of The First International Conference on Autonomous Agents*. The ACM Press, 1997.

[22] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33:159–174, 1977.

[23] R. S. Lockhart. *Introduction to Statistics and Data Analysis for the Behavioral Sciences*. Freeman, 1998.

[24] R. M. Neal. Assessing relevance determination methods using DELVE. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, pages 97–129. Springer Verlag, 1998. See also `http://www.cs.utoronto.ca/~delve/`.

[25] U. Nehmzow, M. Recce, and D. Bisset. Towards intelligent mobile robots - scientific methods in mobile robotics. Technical Report UMCS-97-9-1, University of Manchester Computer Science, 1997. Edited collection of papers, see also related special issue of *Journal of Robotics and Autonomous Systems*, in preperation.

[26] David L. Parnas. Software aspects of strategic defense systems. *American Scientist*, 73(5):432–440, 1985. revised version of UVic Report No. DCS-47-IR.

[27] Phoebe Sengers. Do the thing right: An architecture for action expression. In Katia P Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 24–31. ACM Press, 1998.

[28] Aaron Sloman and Brian Logan. Building cognitively rich agents using the Sim_agent toolkit. *Communications of the Association of Computing Machinery*, 42(3):71–77, March 1999.

[29] L. A. Stein. Challenging the computational metaphor: Implications for how we think. *Cybernetics and Systems*, 30(6):473–507, 1999.

[30] Kristinn R. Thórisson. *Communicative Humanoids: A Computational Model of Psychosocial Dialogue Skills*. PhD thesis, MIT Media Laboratory, September 1996.

[31] Toby Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, 1993. Centre for Cognitive Science.